VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZLEPŠOVÁNÍ SYSTÉMU PRO AUTOMATICKÉ HRANÍ HRY STARCRAFT II V PROSTŘEDÍ PYSC2

IMPROVING BOTS PLAYING STARCRAFT II GAME IN PYSC2 ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAN KRUŠINA**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/12481/2017/xkrusi00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Krušina Jan, Bc.**
Obor: Informační systémy
Téma: **Zlepšování systému pro automatické hraní hry Starcraft II v prostředí PySC2**
Improving Bots Playing Starcraft II Game in PySC2 Environment
Kategorie: Web

Pokyny:

1. Seznamte se s prostředím DeepMind PySC2, umožňujícím snadné učení automatických systémů pro hraní hry Starcraft II.
2. Zpracujte přehled metod pro zlepšování herních strategií pomocí metod strojového učení.
3. Implementujte systém pro zlepšování hry automatického systému na základě učení ze záznamů her.
4. Vyhodnoťte vytvořený systém srovnáním s jinými automatickými systémy, případně lidskými hráči.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- Manning, C. D., Schütze, H., Foundations of Statistical Natural Language Processing, MIT Press, 1999, ISBN 0-262-13360-1.

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá vytvořením automatického systému pro hraní strategické hry v reálném čase Starcraft II. Model je trénován ze záznamů her hráčů a dále využívá technik posilovaného učení pro zlepšování vnitřního systému bota. Záměr je vytvořit systém schopný hrát hru jako celek, přičemž staví na frameworku PySC2 pro strojové učení. Vytvořený bot je poté testován proti skriptovaným botům ve hře.

Abstract

The aim of this thesis is to create an automated system for playing a real-time strategy game Starcraft II. Learning from replays via supervised learning and reinforcement learning techniques are used for improving bot's behavior. The proposed system should be capable of playing the whole game utilizing PySC2 framework for machine learning. Performance of the bot is evaluated against the built-in scripted AI in the game.

Klíčová slova

posilované učení, hierarchické posilované učení, Starcraft II, PySC2, hluboké neuronové sítě, DeepMind, RTS, strategické hry v reálném čase, učení s učitelem, Tensorflow, hluboké učení

Keywords

reinforcement learning, hierarchical reinforcement learning, Starcraft II, PySC2, deep neural networks, DeepMind, RTS, real-time strategy games, supervised learning, Tensorflow, deep learning

Citace

KRUŠINA, Jan. *Zlepšování systému pro automatické hraní hry Starcraft II v prostředí PySC2*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Zlepšování systému pro automatické hraní hry Starcraft II v prostředí PySC2

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Krušina
22. května 2018

Obsah

Seznam obrázků

Kapitola 1

Úvod

Strategické hry představovaly vždy velkou výzvu pro umělou inteligenci. Pokusy hrát tyto hry pomocí počítačových programů sahají desítky let do historie. Ať již se jednalo o jednodušší nebo složitější druhy her, např. dáma nebo šachy, vždy představovaly jakýsi strop pro umělou inteligenci.

Velkého milníku na poli strategických her se podařilo dosáhnout vědcům z DeepMind[1]. Jejich algoritmus AlphaGo si podrobil klasickou deskovou strategickou hru Go a nedávno také šachy a Shogi. Využil k tomu techniky strojového učení, načež se o tuto oblast zvednul nevídaný zájem. Kromě deskových strategických her se DeepMind souběžně zaměřil i na počítačové strategické hry, konkrétně na Starcraft II. Jedná se o pokračování veleúspěšné počítačové hry, jejíž první díl je už roky hojně zkoumán na univerzitách z hlediska využití umělé inteligence pro hraní. Počítačové hry však zatím odolávají pokusům o jejich podmanění, patrně především kvůli své komplexnosti a nutnosti hierarchického smýšlení při hraní.

DeepMind proto uvolnil, ve spolupráci s autory hry, prostředí PySC2, které poskytuje přístup ke hře a umožňuje programům jeho prostřednictvím se hrou komunikovat. Jedná se o otevřenou platformu, naprogramovanou v jazyce Python, která dovoluje trénování modelů umělé inteligence pro hraní této hry.

Cílem této práce je navrhnout automatický systém (program), tzv. bota, pro hraní Starcraftu II pomocí technik strojového učení. Implementace bota probíhá právě v prostředí PySC2 a staví tak na dosavadní práci vědců z DeepMind, se kterou se také srovnává. Bot se zaměřuje na hraní celé hry jako takové, byť zjednodušené. Nesoustředí se tedy pouze na jeden aspekt, např. souboje, ale snaží se o využití všech nezbytných prvků hry, aby byl schopen sehrávat celé zápasy a konkurovat tak již zabudovaným a skriptovaným botům ve hře.

Práce je členěna do sedmi kapitol. Začíná úvodem do problematiky strategických her a motivací k jejich řešení. Druhá kapitola rozebírá základní principy strategických počítačových her, jejich typické prvky a také případná omezení. Ve třetí kapitole budou popsány základní přístupy ke hraní Starcraftu, prvního i druhého dílu, a také bude nastíněna předchozí práce týmu DeepMind a jejich významný posun v hraní (deskových) strategií. Čtvrtá kapitola se zabývá popisem technik, které jsou běžně využívány při vytváření umělé inteligence, a které byly použity v této práci. V návaznosti pokračuje pátá kapitola, která prezentuje implementaci již konkrétně použitých technik. Předposlední, šestá, kapitola po-

[1] DeepMind Technologies je dceřiná společnost skupiny Alphabet Inc., pod kterou spadá také Google.

pisuje experimenty a testování, které bylo provedeno. Sedmou kapitolu pak tvoří závěrečné zhodnocení práce.

Kapitola 2

Strategické hry

Hry obecně jsou ideální platformou pro vývoj a testování umělé inteligence, jelikož poskytují určité omezené herní prostředí a pevně stanovená pravidla. Prvotní úspěchy umělé inteligence v hraní her (deskových) se objevovaly již od 70. let minulého století. Byly schopny porážet lidské protivníky a hrát perfektně různé deskové hry, např. vrhcáby nebo scrabble. Programy postupně získaly schopnost excelovat i ve strategických deskových hrách, např. v dámě nebo šachách [27].

V posledních několika letech se začal výzkum umělé inteligence zaměřovat na strategické počítačové hry. Vědci doufají, že by jim řešení komplexního herního prostředí umožnilo přenést tuto znalost i do ostatních problematik, potažmo přímo do reálného světa, např. robotiky nebo armády [5]. Strategické počítačové hry mají různá omezení, která zabraňují hráčům, aby dokonale promýšleli svoje tahy, např. se odehrávají v reálném čase, tudíž hráč nemá dostatek času na rozmýšlení, k jejich hraní je třeba hierarchického smýšlení a v neposlední řadě hráč nemá kompletní přehled o soupeři, o jeho pozici, infrastruktuře nebo tazích. Zmíněný nedostatek času pro výpočty i neznalost okolí jsou typické problémy, které je nutné řešit v reálném prostředí. Hry tak slouží jako jakýsi simulátor podmínek. Pokud by se podařilo generalizovat řešení takovýchto problémů, které by se dalo využít i v jiných oblastech, jednalo by se o velký pokrok ve vývoji umělé inteligence.

Tato kapitola se zabývá úvodem do problematiky strategických her, především počítačových, a vymezením jejich hlavních prvků a principů.

2.1 RTS

Jedním z druhů strategických počítačových her je RTS (angl. Real-Time Strategy Games). Jedná se o typ strategických her, které se odehrávají v reálném čase. V klasických strategiích (herních i deskových) se hráči typicky střídají v tazích každé herní kolo. V případě RTS hrají oba hráči zároveň a mají tak minimum času na promyšlení svých tahů a strategie. Mezi nejznámější RTS počítačové strategie patří herní série Starcraft, Warcraft, Dune, Command and Conquer nebo Age of Empires. RTS přinášejí několik typických odlišností oproti ostatním typům strategických her:

- *Reálný čas* – hra běží v reálném čase a hráči tak mají omezený čas na rozmýšlení svého tahu nebo promyšlení strategie či příštího postupu.
- *Jednotky* – hráč kontroluje desítky, případně stovky, jednotek naráz na mapě. Musí být tedy schopen s nimi efektivně manipulovat.

- *Budování* – pro výrobu jednotek je zapotřebí stavět vhodné budovy. Je nutné, aby měl hráč dopředu naplánovanou nějakou strategii, které se bude držet, případně změnit pořadí stavění budov tak, aby se přizpůsobil nepříteli.
- *Suroviny* – pro produkci jednotek či výstavbu budov je třeba těžit suroviny, kterých je omezené množství, což typicky nutí hráče k expanzi a stavění dalších základen.
- *Neviditelnost soupeře* – hráč nemá přehled o soupeřových akcích, tedy o jeho jednotkách či budovách, pokud jej sám explicitně neodhalí. Hráči tak musí aktivně zkoumat svoje okolí a nepřítele pomocí svých jednotek (typicky označováno jako skauting).
- *FoW* – tento pojem souvisí s neviditelností soupeře. FoW (angl. Fog of War) je „mlha“, která pokrývá celou herní mapu. Hráč má odkryté jen oblasti, na kterých má svoje vlastní jednotky a zbytek mapy mu zůstává zahalen.
- *Více obrazovek* – většina RTS her má k dispozici dvě obrazovky pro hraní, přičemž je nutné sledovat obě.

Všechny tyto aspekty je nutné brát v potaz a umět s nimi nakládat. To vede hráče k hierarchickému chování, které je nutné si osvojit, aby bylo možné hru dostatečně dobře hrát. Každý zápas typicky začíná umístěním hráčů na jeden z protilehlých rohů mapy. Hráč musí být schopen zajistit dostatečnou produkci surovin pro rozšiřování svého území, zabrat důležitá místa na mapě, špehovat nepřítele, aby se mohl přizpůsobit jeho stylu hry (resp. vhodně se mu bránit), produkovat jednotky a útočit na nepřítele.

Tyto akce se dělí na dvě skupiny – mikromanagement (dále jen mikro) a makromanagement (dále jen makro), což jsou typické pojmy pro Starcraft. Makrem se dá označit dlouhodobé strategické plánování. Především pořadí stavění budov, vylepšení technologií a zvolení vhodné produkce jednotek. Správné naplánování těchto akcí je nezbytné pro dostatečnou obranu proti nepříteli. Jejich vykonávání má pak dlouhodobé následky ve hře (např. hráč se musí adekvátně přizpůsobit nepříteli, který hraje agresivně a naopak). Opakem makra je mikro. Mikro je souhrnný název pro efektivní využití jednotek v boji. Hráč musí být schopen efektivně koordinovat svoje vojáky, aby mohl útočit na nepřítele. Vhodným mikrem je možné porazit silnější jednotky slabšími (např. kličkováním či jinou taktikou). Zkušení hráči typicky koordinují různé skupiny vojáků současně, aby je co nejefektivněji využili.

RTS hry jsou tedy kombinací dlouhodobých i krátkodobých cílů. Typicky obsahují několik různých ras, za které je možné hrát. Každá z nich má desítky různých typů budov, jednotek a vylepšení, které je možné využít, což vede k ohromnému množství možných akcí a stavů.

2.2 Starcraft II jako RTS

Starcraft II je RTS počítačová hra vydaná v roce 2010 společností Blizzard Entertainment[2]. Řadí se do žánru sci-fi her. Jedná o pokračování prvního dílu z roku 1998 od stejné společnosti. První i druhý díl si získal celosvětově velkou oblibu a patří dodnes mezi nejlepší (strategické) počítačové hry, které kdy byly vytvořeny [18] [19].

Starcraft se těší popularitě po celém světě, ale patrně nejpopulárnější je v Jižní Koreji, kde se pravidelně odehrávají turnaje o výhry v hodnotě milionů dolarů [3]. Vzhledem

[2]Blizzard Entertainment je americká herní společnost založená v roce 1991.

k množství celosvětově pořádaných šampionátů a profesionálních hráčů, kteří hru hrají, se hra často označuje jako tzv. e-sport (elektronický sport).

Samotná hra obsahuje tři rasy, za které může hráč hrát, jedná se o Terrany, Zergy a Protosse. Každá rasa nabízí svůj unikátní styl hraní a přispívá tak k celkové komplexnosti a různorodosti hry. Pro každou rasu jsou k dispozici téměř dvě desítky unikátních vojenských jednotek, se kterými může hráč manipulovat, přičemž každá z nich je vhodná v odlišné situaci. Některé jednotky mají výhodu v boji ve vzduchu, případně na zemi, jiné jsou univerzální. Hráči tak musí mít přehled o výhodách a nevýhodách svých a protivníkových jednotek a vhodně je kontrovat. Dále má každá rasa kolem desítky různých budov, které produkují zmíněné jednotky či vyvíjí vylepšení pro již stávající jednotky. Tato vylepšení poskytují výhodu vojenským jednotkám, protože zlepšují některé jejich statistiky, např. počet životů, štít nebo silnější útok. Jednotlivá vylepšení bývají drahá, takže hráč musí brát v potaz, kdy se jaké vylepšení hodí vyzkoumat, a kdy by mu naopak úbytek surovin mohl způsobit potíže.

Pro ovládání hry má hráč k dispozici dvě hrací plochy – obrazovku a minimapu (viz obr. 2.1). Minimapa znázorňuje celou herní mapu ve zmenšené podobě a s nedostatečně detailními informacemi. Obsahuje prakticky pouze informace o rozmístění objevených budov a jednotek (vlastních i nepřátelských). Obrazovka pak představuje zvětšený výřez z minimapy, který je možné libovolně změnit pomocí posouvání kamery po minimapě. Obrazovka obsahuje detailní pohled na zvolenou část herní mapy. Je tedy možné vidět kromě jednotek a budov i jejich konkrétní stav, tzn. například směr střelby vojáků, jejich přesný pohyb, zbývající životy apod. Pro lepší úroveň hraní se hráč musí soustředit na obě hrací plochy, protože kdyby se zaměřil jen na jednu z nich, tak by byl značně znevýhodněn.

Hráči začínají na protilehlých stranách mapy s několika těžebními jednotkami a jednou budovou (velitelským centrem). Musí těžit suroviny pro výrobu jednotek a stavění budov. Zajištění nepřetržitého přísunu surovin je stěžejní úkol po celou dobu zápasu. Suroviny jsou strategicky rozmístěny po mapě a je jich pouze omezené vytěžitelné množství. Hráč tak musí postupně zvětšovat svoje území, aby měl zajištěn přísun dostatečného množství materiálu. Předsunuté základny je nutné bránit před protivníkem, který by jinak mohl snadno zničit nechráněné těžící jednotky a přísun surovin zastavit.

Starcraft je hra s nedokonalou informací, tzn. ani jeden z hráčů nemá kompletní přehled, respektive téměř žádný přehled, o tom druhém, pokud ho aktivně nešpehuje. Hra obsahuje FoW, která odkrývá pouze tu část herní mapy, na které se nacházejí hráčovy vlastní jednotky. Území, které hráč vůbec neprozkoumal, je zcela zahaleno a území, na kterém nemá momentálně svoje jednotky, je taktéž překryto, i když bylo předtím prozkoumáno. Na takto překrytém území je vidět pouze terén a hráč se zde musí aktivně pohybovat, aby nad ním měl přehled. Typicky se tak v ranné fázi hry využívá základních těžebních jednotek, které aktivně prozkoumávají mapu a hledají, kde se nachází nepřátelská základna. Po jejím nalezení je často cíl těchto jednotek pohybovat se v její blízkosti dostatečně dlouhou dobu na to, aby odhalily možnou zvolenou strategii hráče podle toho, jaké budovy začíná stavět jako první. V terminologii Starcraftu se posloupnost stavění budov typicky označuje jako tzv. build-order, jenž je předzvěstí pravděpodobně zvolené některé ze známých taktik (např. typická defenzivní nebo ofenzivní strategie). Špehování nepřítele se provádí především s cílem adekvátně se přizpůsobit jeho zvolené strategii.

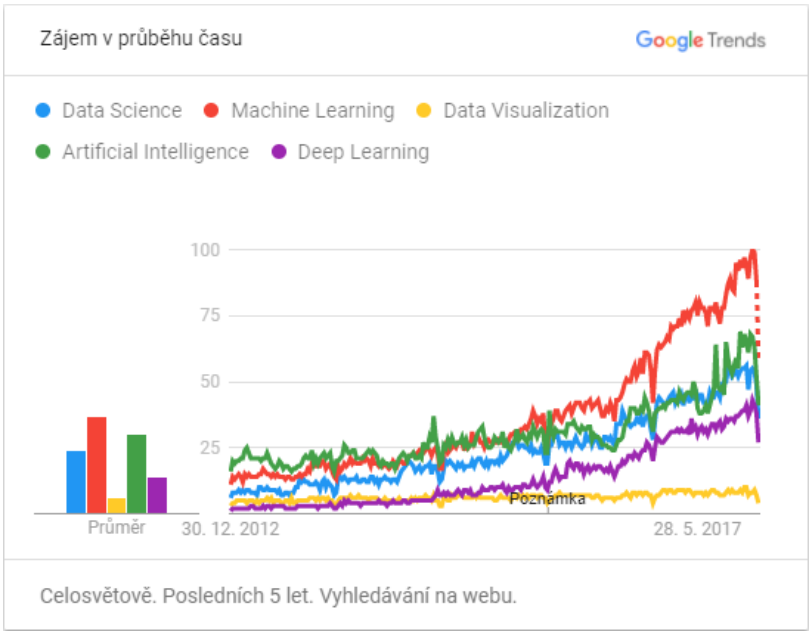
Dle zvolené strategie a stylu hraní se pak délka jednoho zápasu v soubojích 1vs1 pohybuje mezi 10 až 20 minutami. Při agresivních stylech hraní se tato doba pohybuje mezi 5 až 10 minutami. Naopak u defenzivního stylu to může být výrazně více než 20 minut.

Obrázek 2.1: Ukázka uživatelského rozhraní Starcraftu II. Jedná o snímek obrazovky, na kterém je vidět hráčská základna za rasu Terran. Základna je tvořena několika budovami a vpravo nahoře se nachází těžební jednotky, které zajišťují přísun surovin. V levém dolním rohu je pak minimapa, ze které je patrné, že většina herní mapy je stále zakryta pomocí FoW.

Kapitola 3

Hraní her pomocí umělé inteligence

Oblast umělé inteligence se v posledních letech těší velkému zájmu, čemuž odpovídá i statistika počtu dotazování na vyhledávácím enginu Googlu na termíny spjaté se strojovým učením, viz obr. 3.1. Této popularitě přispěly i nedávné úspěchy týmu DeepMind, kterému se podařilo docílit několika milníků v oblasti umělé inteligence. Jedná se například o dosáhnutí expertního stylu hraní deskových strategických her [31], či klasických počítačových her na Atari [21].



Obrázek 3.1: Statistika vyhledávání termínů spjatých se strojovým učením. Osa y znázorňuje relativní zájem ve vyhledávání slovních spojení skrze Google, kde hodnota 100 značí nejvyšší míru zájmu a 0 nejnižší. Je patrné, že zájem v roce 2017 několikanásobně vzrostl oproti předchozím rokům [16].

První část kapitoly bude věnována úvodu do hraní strategických her obecně, a to především v souvislosti s programem AlphaGo, který odstartoval dnešní zájem o umělou inteligenci ve hrách, úspěch týmu DeepMind a který také ovládnul hraní klasické deskové

strategické hry Go. Pozornost bude věnována prvotnímu algoritmu i jeho novějším variantám. Tato zmínka byla vybrána pro lepší zasazení prací týmu do kontextu strategických her jako takových.

Navazující části se zabývají rozborem principů některých existujících systémů, jak pro první díl Starcraftu, tak i pro druhý, od základních skriptovaných botů po boty zlepšující se pomocí metod strojového učení.

Jelikož API ke druhému dílu Starcraftu bylo zpřístupněno teprve současně s vydáním práce DeepMindu, neexistují doposud žádné komplexnější automatizované systémy, dle vědomí autora této práce, založené na strojovém učení, které by hru hrály. Z hlediska Starcraftu II tak bude popsán v této části pouze bot od týmu DeepMind.

3.1 Strategické hry

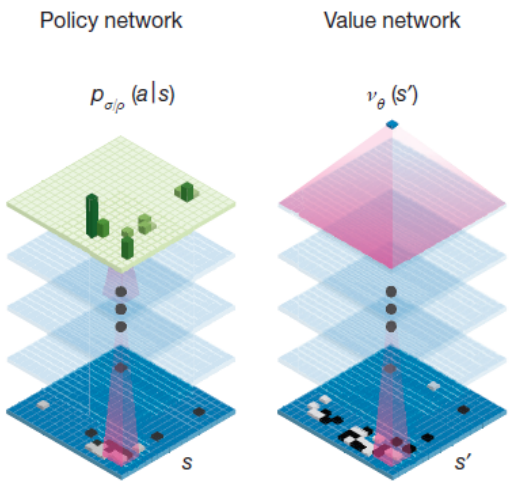
První významný úspěch týmu vědců z DeepMind v čele s D. Silverem na poli strategických her se podařil v roce 2016, kdy jejich algoritmus označovaný jako AlphaGo porazil šampiona ve hře Go skórem 5:0 a s úspěšností 99,8 % přehrál všechny ostatní počítačové programy [30]. Dnes je tento algoritmus schopný excelovat i v dalších hrách jako jsou šachy nebo Shogi.

Obrázek 3.2: Go, Shogi a šachy. Prastará trojice klasických deskových strategických her, které jsou dodnes populární, a ve kterých excelují algoritmy z rodiny Alpha [37] [38] [36].

3.1.1 AlphaGo a jeho varianty

Go je stará strategická desková hra, která pochází z Číny. Obvykle se hraje na desce o rozměrech 19x19 polí s černými a bílými kameny, které hráči střídavě kladou na průsečíky na desce, přičemž jako první začíná hráč s černými kameny. Cílem hry je získat kontrolu nad co největším územím pomocí kamenů, ať již kladením svých kamenů na desku, či zabíráním těch soupeřových [17].

Tato hra se řadí mezi hry s dokonalou informací. Hráč tak má kompletní přehled o tazích soupeře a všech předchozích událostech. U her tohoto typu je dostupná optimální funkce ohodnocení, která v každém stavu s udává výsledek dané hry, za předpokladu, že hráč hraje racionálně a volí pro něj nejlepší možný tah, dané hry. Go se vyznačuje rozsáhlým počtem všech možných akcí, které je možné provést. Odhad proveditelných akcí je dán vztahem b^d, kde b je počet všech možných korektních akcí ($b \approx 250$), a d je délka hry ($d \approx 150$), tedy přibližně 10^{360} možných kombinací (na rozdíl od šachů, kde je to přibližně 10^{123} možných kombinací) [30].



Obrázek 3.3: Architektura neuronových sítí v AlphaGo. Vstup pro obě neuronové sítě je tvořen reprezentací hrací desky. Síť strategie produkuje rozdělení pravděpodobnosti tahů a síť ohodnocení počítá funkci ohodnocení pro daný stav [30].

Cílem AlphaGo bylo tedy zredukovat možnou šířku a hloubku prohledávání na snesitelnou úroveň. K tomuto účelu byly využity dvě hluboké neuronové sítě, jedna pro zvolení strategie hraní, a druhá pro výpočet funkce ohodnocení (obr. 3.3). Síť byla natrénována z 30 milionů různých pozic ve hře a dosahovala úspěšnosti předpovědi dalšího tahu hráče až 57 %. Další fáze učení proběhla za využití posilovaného učení, s jehož pomocí předpovídá nejlepší možné tahy pro zajištění výhry. V této fázi sehrálo AlphaGo více než milion her samo se sebou, přičemž fáze jednotlivých her byly zvoleny náhodně, aby se předešlo přetrénování sítě. Pomocí posilovaného učení bylo AlphaGo schopno vyhrát více než 80 % všech her, které hrálo.

Algoritmus efektivně kombinuje vyhodnocování strategie a ohodnocení pomocí neuronových sítí s vyhledávacími stromy Monte Carlo.

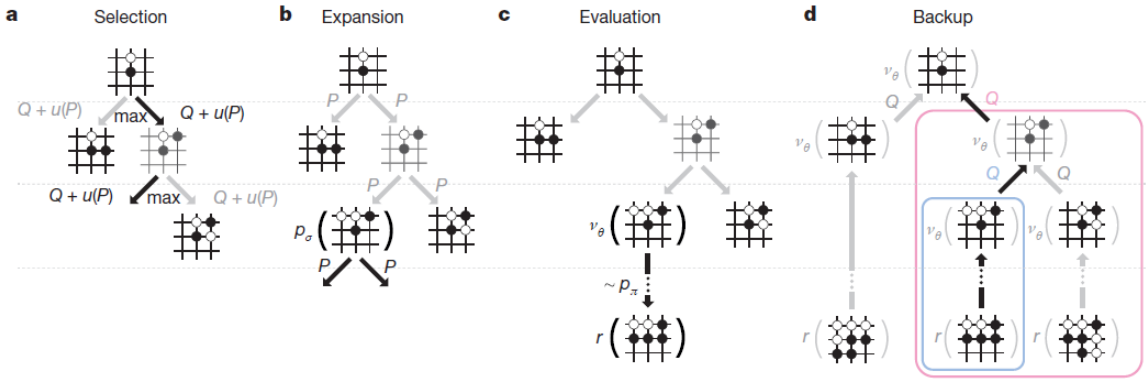
AlphaGo bylo schopno porazit evropského velmistra F. Huie skórem 5:0 ve hře bez handicapů, tzn. hrálo na hrací desce v obvyklé velikosti. Později porazilo i jihokorejského šampiona L. Sedola skórem 1:4.

Druhá verze AlphaGo, nazvaná AlphaGo Zero, se liší od svého předchůdce tím, že využívá čistě posilovaného učení pro zlepšování hraní. V předchozí verzi byly použité neuronové sítě trénovány jak skrze učení s učitelem, tak i skrze hraní proti sobě pomocí posilovaného učení. AlphaGo Zero je schopna učit se sama bez jakékoliv předchozí znalosti hry, pouze skrze herní pravidla.

Zero využívá jednu neuronovou síť f_θ, kde θ jsou parametry sítě, jejíž vstup tvoří reprezentace hrací desky, a výstup tvoří funkce ohodnocení v a vektor rozdělení pravděpodobnosti tahů $\boldsymbol{p}$:

$$(\boldsymbol{p}, v) = f_\theta \tag{3.1}$$

Síť tedy slouží pro predikci strategie i ohodnocení. Na rozdíl od předchozí verze, kde byly použity sítě dvě. Neuronová síť je natrénována skrze hry, které hrála sama proti sobě, k čemuž využívá vyhledávání Monte Carlo (angl. Monte Carlo Tree Search), ukázka na obr. 3.4. MCTS prohledává akce, které předpovídá síť f_θ, což umožňuje vybírat potenciálně



Obrázek 3.4: Prohledávací strom Monte Carlo. Průchod stromem je rozdělen do čtyř fází. Nejprve dochází k vybrání uzlu, do kterého vede hrana s největším ohodnocením Q a s apriorní pravděpodobností. Pokud je daný uzel expandován, tak dochází k jeho vyhodnocení sítí a jsou provedeny rozehrávky daného uzlu. Q hodnoty si pak ukládají průměrné hodnoty vyhodnocených uzlů [30].

lepší akce, než které předpovídá samotná síť. MCTS tak slouží jako prostředník pro zlepšení strategie.

Program v každém uzlu (stavu) expanduje jeho synovské uzly a ohodnotí každý z nových uzlů. V každém novém uzlu sehraje hru a uzlu přičte hodnotu $+1/0/-1$ podle výhry, remízy nebo prohry. Algoritmus uchovává dvě hodnoty pro každý uzel, W, která značí celkovou hodnotu ohodnocení pro daný uzel a N, která značí počet rozehrávek, které byly v daném uzlu, nebo jeho synovských uzlech, provedeny. Ze synovských uzlů je tedy jejich ohodnocení propagováno zpět k rodičům. MCTS neexpanduje všechny listové uzly, ale pouze ty, které mají určité skóre podle UCT (angl. Upper Confidence Bound Trees) U, to je definováno následovně:

$$U_i = \frac{W_i}{N_i} + c \cdot \sqrt{\frac{\ln N_p}{N_i}} \tag{3.2}$$

Kde W_i je naakumulovaná hodnota synovského uzlu, N_i je počet navštívení uzlu, N_p je počet navštívení rodičovského uzlu a $c \geq 0$ je parametr, který udává kompromis mezi navštěvováním čistě nejvýše ohodnoceného uzlu a navštívením málo ohodnoceného uzlu (takový uzel nemusel být ještě patřičně prozkoumám a mohl by potenciálně vést k lepšímu ohodnocení).

AlphaGo Zero porazilo svého předchůdce ve sto hrách čistým skóre 100:0.

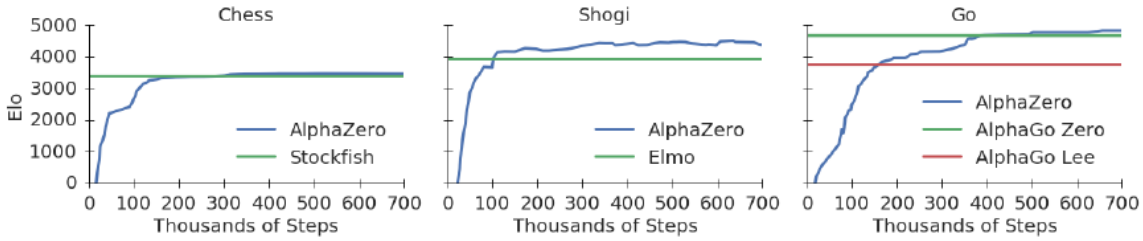
Nejnovější typ algoritmu AlphaGo z dílny DeepMind nese název AlphaZero a byl schopen si osvojit nadlidskou úroveň hraní v šachách, shogi i Go, viz tabulka 3.5. Jedná o zobecněný algoritmus, který navazuje na předešlé, nevyužívá žádnou předchozí znalost hry, kromě pravidel, a neučí se ze záznamů her profesionálních hráčů. Ve všech třech zmíněných hrách byl schopen porazit světové šampiony i všechny ostatní programy, zabývající se hraním těchto her.

Snaha podrobit si šachy panuje již desetiletí. Existuje spousta teorií a analýz, jak hru hrát. Pro výzkumníky a umělou inteligenci to byla vždy velká výzva. Prvního významného úspěchu se podařilo dosáhnout v roce 1997, kdy superpočítač Deep Blue porazil tehdejšího velmistra v šachu G. Kasparova [6]. Programy se poté nadále rozvíjely, nicméně jejich pod-

stata spočívala v propočítávání pozic, které byly zadávány a ručně upravovány šachovými mistry s využitím např. alfa-beta prohledávání a speciálních heuristik. Mezi nejlepší takové systémy se doposud řadil program Stockfish [26].

Stěžejní prvky algoritmu se příliš neliší od AlphaGo Zero, AlphaZero opět využívá jednu neuronovou síť pro získání ohodnocení a pravděpodobností tahů a MCTS pro zlepšení výběru možných tahů. Liší se pouze v několika věcech. Předchozí verze využívaly bayesovskou optimalizaci pro upravování hyperparametrů, AlphaZero používá pro všechny hry stejné, které dále neoptimalizuje. Pro každou hru byl však algoritmus natrénován zvlášť.

AlphaZero bylo schopno porazit doposud nejlepší program Stockfish v šachách za čtyři hodiny trénování. V Shogi porazilo program Elmo za méně než dvě hodiny a v Go předčilo předchozí verzi po osmi hodinách tréninku.



Obrázek 3.5: Srovnání dosaženého Ela ve strategických hrách. Jednotlivé grafy znázorňují dosažené Elo během trénování algoritmů z rodiny Alpha a případně dalších systémů [31].

3.2 Starcraft

Vyvíjené automatické systémy se dají rozdělit na dvě skupiny, na ty systémy, které jsou striktně skriptované a využívají ke hraní pouze předem naprogramované vzorce chování, a na ty, které se snaží pomocí různých technik strojového učení hrát hru nebo nějakou její část bez předem připravených pravidel chování. V této podkapitole budou představeni zástupci pro první díl Starcraftu z obou těchto kategorií.

3.2.1 Skriptované systémy

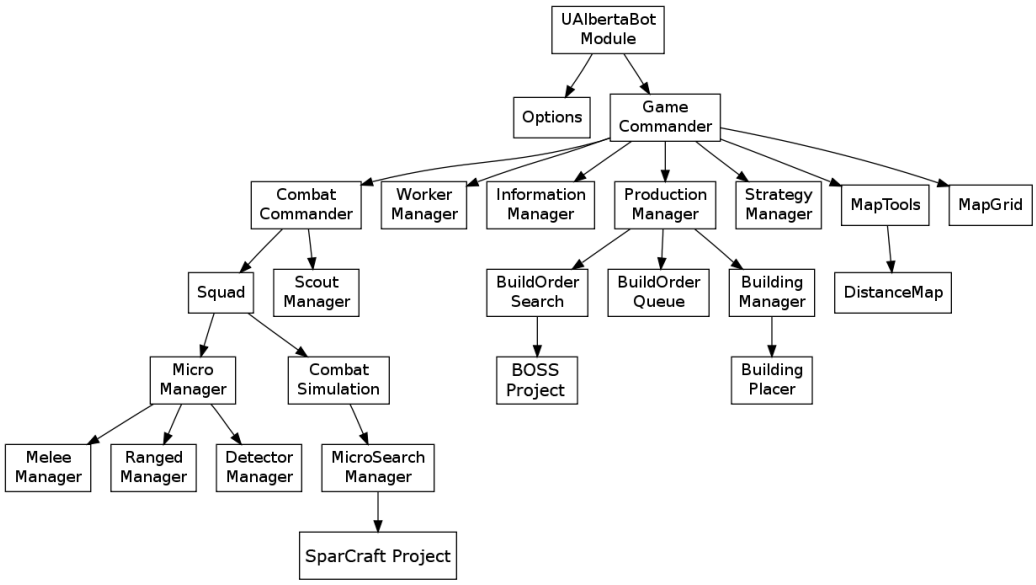
Po zveřejnění API pro první díl Starcraftu v roce 2009 (nazývané BWAPI[3]), které umožňovalo komunikaci s herním enginem, se začali výzkumníci i nadšenci zajímat o Starcraft z hlediska umělé inteligence. Díky BWAPI bylo možné vytvářet automatické systémy, které by hru hrály. Vzhledem k tomu, že existovala i podpora multiplayeru, tedy hry více hráčů, bylo možné boty srovnávat s jinými boty či živými hráči.

To vedlo k zavedení turnajů mezi automatickými systémy, které mezi sebou soupeřily. Mezi nejznámější soutěže, které se stále konají patří např. AIIDE[4], SSCAIT[5] či CIG[6]. Turnaje se konají většinou pod záštitou univerzit, takže se do nich obvykle zapojují i studenti.

Hraní RTS her je pro umělou inteligenci velmi náročné, jak již bylo zmíněno v úvodu. Vyžaduje kombinaci několika činností, především makromanagementu, mikromanagementu a strategie. Systémy tak nedosahují prozatím takové úrovně, aby byly schopny porazit lidské

[3] https://bwapi.github.io/
[4] https://sites.google.com/view/aiide2017/
[5] https://sscaitournament.com/
[6] https://cilab.sejong.ac.kr/sc_competition/

hráče. Boti byli však doposud pouze z většiny skriptováni (tedy obsahují napevno zakódovaná pravidla a strategie, dle kterých se chování, a které se spustí v určitých situacích).



Obrázek 3.6: Typická architektura skriptovaných botů. Logika bota je hierarchicky členěna na moduly, z nichž jsou nejdůležitější moduly starající se o armádu, sbírání surovin, zkoumání mapy a strategii [9].

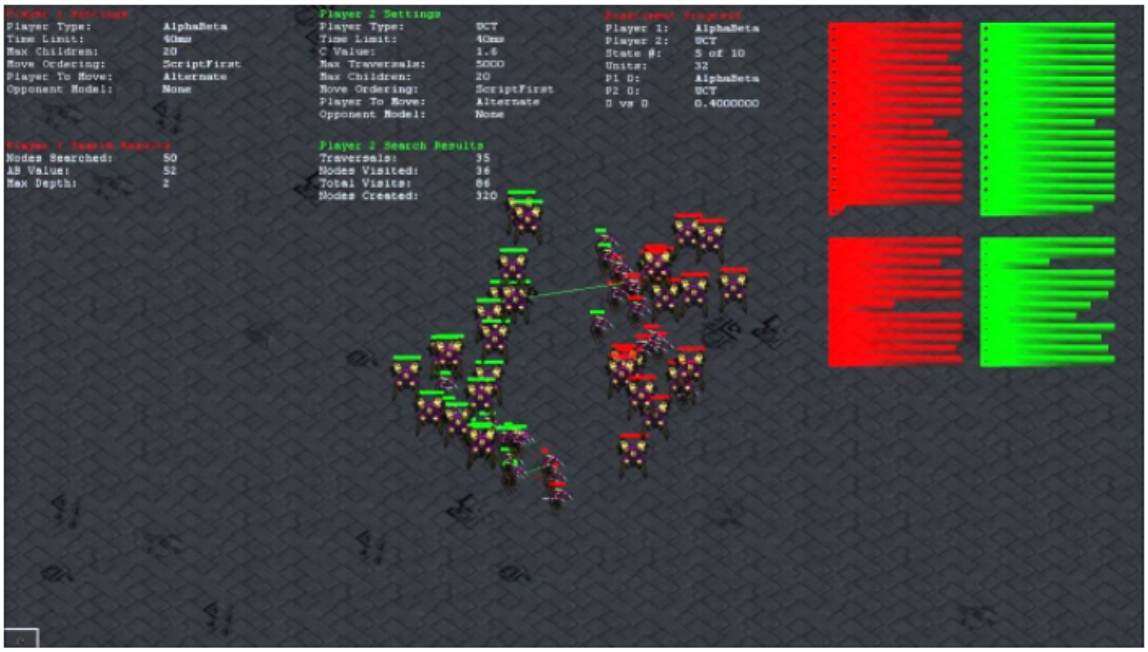
Patrně nejznámější bot pro Starcraft je automatický systém od D. Churchilla nazývaný CommandCenter [7]. Jeho základní implementace existuje již i pro druhý díl Starcraftu. Program je psaný v jazyce C++ a komunikuje se hrou prostřednictvím BWAPI.

Jedná se především o skriptovaného bota. Jeho architekturu tvoří několik manažerů (agentů), kteří mezi sebou komunikují a reprezentují chování bota. Typická architektura těchto systémů je vidět na obr. 3.6. Program staví na základech původního UAlbertaBota [9]. Ten využívá simulátor SparCraft (viz obr. 3.7), což je modul, který umožňuje nasimulovat předem různé soubojové situace a výsledky z nich pak využít pro rozhodování se. SparCraft využívá jak některé předem definované a naskriptované situace, tak i vyhledávací algoritmus ABCD, což je upravený alfa-beta algoritmus, který umožňuje heuristické stromové vyhledávání skrze možné akce [10]. Bot také dále také využívá modul BOSS, ten umožnuje automatizovaně plánovat a sestavovat build-ordery [8].

Bot je složen z multiagentní architektury, přičemž jednotliví agenti jsou logicky a hierarchicky odděleni a každý z nich se stará o jinou část hry. Mezi hlavní moduly patří velitelský modul, který má pod sebou všechny ostatní. Dále jsou to moduly starající se o těžební jednotky, ovládání jejich pohybu a zajištění efektivního těžení, produkční moduly starající se o stavění potřebných budov a vojenské moduly, které ovládají pohyb a útoky všech vojenských jednotek. Mezi další podpůrné moduly patří např. moduly zkoumající mapu, sbírání informací o nepříteli apod.

CommandCenter, resp. UAlbertaBot, slouží typicky jako framework pro mnoho ostatních botů účastnících se soutěží, kteří staví na jeho základních funkcích a individuálně je zlepšují. Mezi takové patří např. LetaBot[7].

[7] https://github.com/MartinRooijackers/LetaBot



Obrázek 3.7: Snímek souboje ze simulátoru SparCraft. SparCraft umožňuje nasimulovat různé soubojové situace a odvodit z nich další chování ve hře [8].

3.2.2 Multiagentní rekurentní boti

Pro hraní Starcraftu je typické, že mezi sebou komunikují multiagentní architektury uvnitř automatického systému. Práce [24] čínských vědců se zaměřuje na jádro bota skrze obousměrné rekurentní neuronové sítě, což jsou sítě, které mohou komunikovat oběma směry. Neurony tak mají dostupné informace jak zpětně, z předešlých stavů, tak i dopředně, z nadcházejících stavů, což zvyšuje propustnost informací. Takové sítě se hodí především v případech, kdy je nutné chápat delší posloupnosti akcí [29].

Komunikace agentů tedy probíhá obousměrně skrze navrženou síť, která byla pojmenována BiCNet (angl. Bidirectionally-Coordinated Network). Skládá se z multiagentní sítě aktérů a multiagentní sítě kritiků (diagram 3.8). Obě dvě sítě, aktér i kritik, jsou založeny na zmíněných rekurentních sítích. Aktér, neuronová síť pro strategii, produkuje výslednou akci pro agenta, kritik, neuronová Q-síť, tento výběr ohodnocuje.

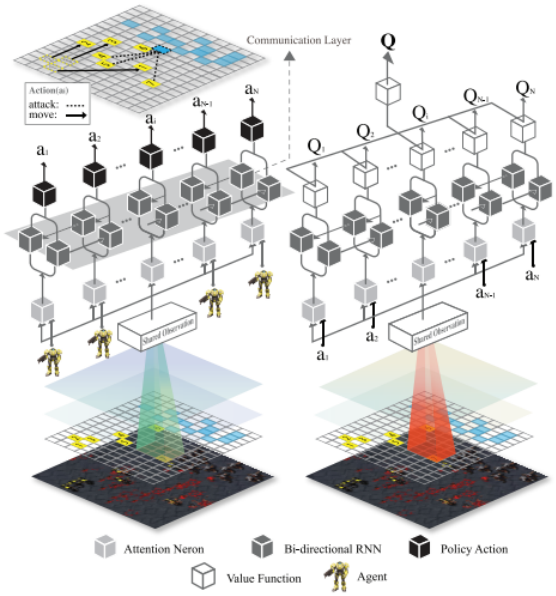
BiCNet zvládá více typů soubojových scénářů při libovolném počtu agentů. Navíc ke svému fungování nevyžaduje žádná expertní lidská data (např. záznamy z her), a učí se tak sama.

Práce se zaměřuje na mikromanagement, tedy koordinaci vojáků v soubojích. V každém scénáři jsou agenti na obou stranách, berou tedy souboj jako hru s nulovým součtem mezi dvěma hráči s N a M jednotkami.

Funkce odměny je navržena jakožto globální, kde výše odměny je rozdíl v životech jednotek mezi dvěma po sobě jdoucími snímky hry:

$$r(s,a,b) = \frac{1}{M}\sum_{j=N+1}^{N+M}\Delta R_t^t(s,a,b) - \frac{1}{N}\sum_{i=1}^{N}\Delta R_t^t(s,a,b) \tag{3.3}$$

Kde t je krok, s je stav, a jsou akce agenta, b akce nepřítele, přičemž $a \in A$, $b \in B$, A, B jsou množiny akcí a $\Delta R_i^t(\cdot)$ je množství ubraných životů jednotek.



Obrázek 3.8: Multiagentní architektura bota. BiCNet spojuje multiagentní sítě strategie a multiagentní Q-sítě pro lepší komunikace mezi agenty [24].

Síť byla testována na několika různých soubojových scénářích, od lehkých po těžké a srovnávána s jinými multiagentními boty (GMEZO, CommNET), skriptovaným botem (IND), či botem implementovaným pomocím plně propojených sítí (FC). Síť měla ze všech nejlepší procentuální úspěšnost v soubojích mezi rasami Terran vs. Zerg s jednotkami Mariňáků proti Zerglingům (což jsou základní jednotky daných ras), více v tabulce 3.9.

Combat	Rule Based			RL Based				
	Built-in	Weakest	Closest	IND	FC	GMEZO	CommNet	BiCNet
20 M vs. 30 Z	**1.00**	.000	.870	.940	.001	.880	**1.00**	**1.00**
5 M vs. 5 M	.720	.900	.700	.310	.080	.910	**.950**	.920
15 M vs. 16 M	.610	.000	.670	.590	.440	.630	.680	**.710**
10 M vs. 13 Z	.550	.230	.410	.522	.430	.570	.440	**.640**
15 W vs. 17 W	.440	.000	.300	.310	.460	.420	.470	**.530**

Obrázek 3.9: Tabulka úspěšnosti BiCNetu v různých scénářích. BiCNet měl nejlepší procentuální úspěšnost v soubojích [24]

3.3 Starcraft II

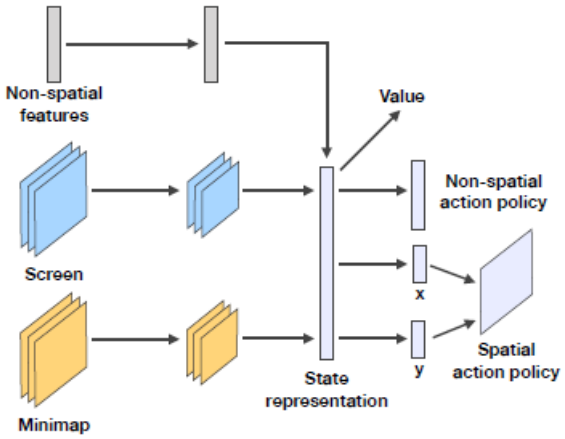
V této části bude podrobněji rozebrán a představen bot týmu DeepMind, který se pomocí strojového učení snaží hrát celou hru.

V současnosti nejsou autorovi této práce známé žádné další práce, které by se zabývaly hraním druhého dílu Starcraftu pomocí agenta založeného na strojovém učení. Všechny dosavadní pokusy se zabývají replikací práce DeepMindu, tj. implementací agenta, který se bude učit hrát přiložené minihry. Existující boti jsou ručně skriptovaní, psaní v jazyce C++ a mají obdobnou strukturu popsanou výše v části 3.2.1.

3.3.1 DeepMind Bot

DeepMind se ve své práci [35] zaměřují na vytvoření bota z pohledu posilovaného učení. Staví při tom na svých předchozích úspěších na poli umělé inteligence při řešení her. Základem je neuronová síť (resp. více sítí, které pak mezi sebou porovnávají), jejíž vstup tvoří snímky hry převedené do tzv. vrstev rysů, přičemž každý z rysů reprezentuje jiné údaje (např. rozmístění jednotek, jejich životy, viditelnost mapy apod.). DeepMind se zaměřují na hru jako celek a nesoustředí se pouze na jednu část hry, jako tomu bylo u většiny dosavadních prací v souvislosti se strojovým učením.

První část jejich neuronové sítě tvoří konvoluční vrstvy, které zpracovávají vstupní snímky. Ty jsou následně spojeny a pomocí plně propojených sítí tvoří výstup sítě funkce ohodnocení, rozdělení pravděpodobnosti pro prostorové souřadnice a rozdělení pravděpodobnosti pro neprostorové akce. Ukázka jedné ze sítí je na obr. 3.10.



Obrázek 3.10: Ukázka jedné z architektur bota. Jedná o architekturu označovanou jako *Atari-Net*, jejím výstupem je funkce ohodnocení, rozdělení pravděpodobnosti pro neprostorové akce a rozdělení pravděpodobnosti pro prostorové souřadnice akce x a y jednotlivě [35].

Neuronová síť se učí pomocí algoritmu A3C, který využívá následující gradient pro optimalizaci:

$$(G_t - v_\theta(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t) + \beta(G_t - v_\theta(s_t))\nabla_\theta v_\theta(s_t) + \eta \sum_a \pi_\theta(a|s) \log \pi_\theta(a|s) \quad (3.4)$$

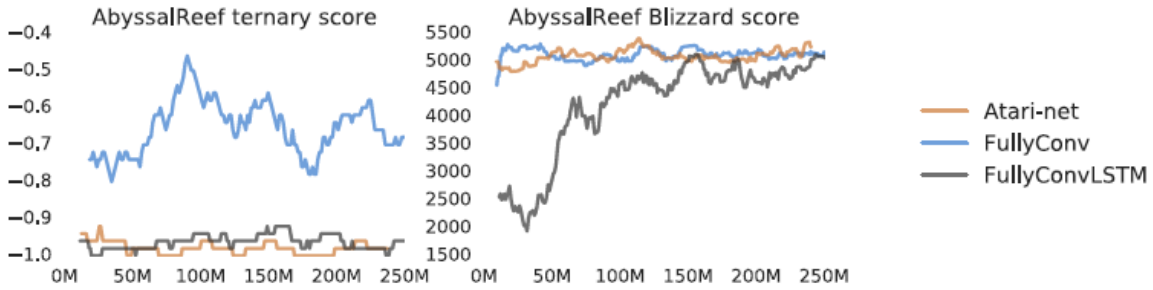
Kde první výraz rovnice je vyjádření gradientu strategie π_θ, druhou část tvoří gradient funkce ohodnocení $v_\theta(s_t)$ a třetí entropie strategie. Dále θ jsou parametry sítě, a je zvolená akce, s je stav, t je krok, G je očekávaná odměna a β, η jsou parametry pro snížení hodnoty příslušného výrazu.

Tým ke své práci uvolnil také několik miniher (celkem 7), jejichž konkrétní popis je možný dohledat v oficiální dokumentaci[8], na kterých testoval svůj algoritmus. Jednotlivé minihry zahrnují například těžení surovin, trénování jednotek, stavění budov nebo jednoduché soubojové operace. Ve většině miniher si vedl dobře a dosáhl excelentního skóre, které předčilo i skóre lidského šampiona, se kterým byl algoritmus srovnáván.

[8] https://github.com/deepmind/pysc2/blob/master/docs/mini_games.md

DeepMind také využilo záznamy z her, které mělo k dispozici. Celkem se jednalo o asi 800 tisíc záznamů zápasů. Na těchto záznamech trénoval síť, která předpovídala výsledek zápasu v průběhu hry. Dle testování byla schopná předpovědět výsledek hry s přesností přes 60 %. Stejnou síť taktéž použili pro předpovídaní lidských akcí. S téměř 38% úspěšností je pak síť schopna odhadnout lidské chování, tedy akci, kterou jedinec zvolí příště. Pokud akce vyžaduje další (prostorové) argumenty, tak je schopna je odhadnout s přesností přes 10 % pro obrazovku a přes 25 % pro minimapu v nejlepších případech.

Při nasazení bota proti zabudované umělé inteligenci ve hře si však bot nebyl schopen vyvinout úspěšnou strategii pro hraní hry. Ani proti nepříteli na nejlehčí úroveň obtížnosti nebyl schopen vyhrát jedinou hru (viz graf 3.11). Je tedy zřejmé, že samotné posilované učení v čisté podobě na tak komplexní hru nestačí.



Obrázek 3.11: Dosažené skóre DeepMind bota ve hře. Graf zobrazuje dosažené skóre (osa y) v plné hře na mapě *Abyssal Reef* srovnáním jednotlivých architektur sítě. Graf vlevo zobrazuje skóre při využití odměn $+1/0/-1$, graf vpravo pak při využití interního skóre hry [35].

Kapitola 4

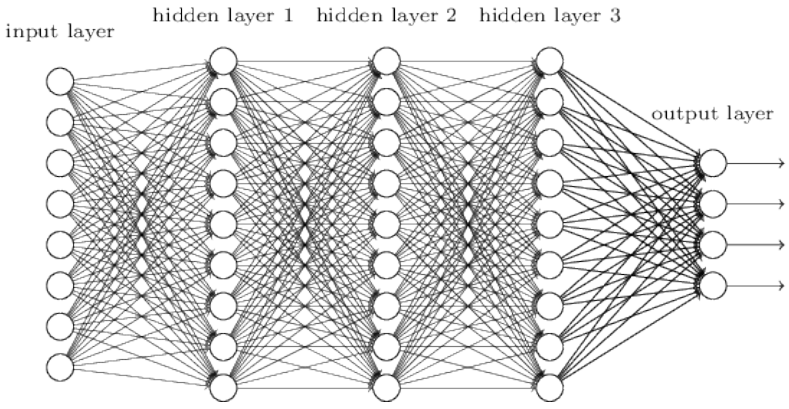
Strojové učení

Následující kapitola rozebírá postupy uplatněné ve strojovém učení, které byly zároveň využity při návrhu prezentovaného automatického systému.

Nejprve budou v krátkosti zmíněny neuronové sítě a princip jejich fungování, dále budou představeny techniky strojového učení, které tvoří základ bota, konkrétně posilované učení a jeho nadstavby v podobě hlubokého a hierarchického posilovaného učení. Tato podkapitola vychází převážně z knihy R. Suttona o posilovaném učení [32]. Poté bude blíže představena a popsána platforma PySC2 pro Starcraft II, pomocí níž se dají trénovat modely agentů skrze komunikaci se hrou, a nakonec bude přiblížena knihovna Tensorflow, která umožňuje implementaci těchto modelů.

4.1 Neuronové sítě

Neuronové sítě jsou základním stavebním kamenem algoritmů strojového učení. Dá se říct, že jsou modelovány tak, aby připomínaly strukturu a chování lidského mozku. Jednoduché sítě se skládají ze tří vrstev, tj. vstupní, skryté a výstupní. Každá vrstva obsahuje určitý počet malých jednotek, neuronů, které zpracovávají vážený vstup a produkují výstup pomocí aktivační funkce. Mezi základní aktivační funkce patří například *sigmoid* nebo *hyperbolický tangens*.



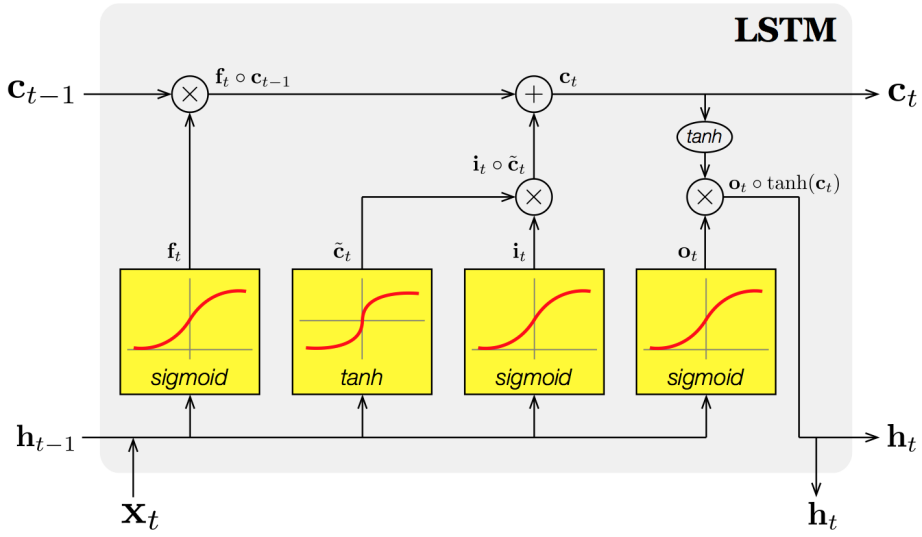
Obrázek 4.1: Ukázka hluboké neuronové sítě skládající se z plně propojených vrstev. Obsahuje vstupní a výstupní vrstvu a tři vrstvy skryté [22].

Pro učení sítě je nezbytné definovat tzv. loss funkci, která určitým způsobem počítá chybu sítě (odchylku od chtěného chování), a její postupnou minimalizací je dosažen proces učení a zlepšování. Obvykle se váhy sítě aktualizují pomocí metody zpětné propagace po vypočítání gradientů loss funkce.

V současnosti se pak v praxi používají především hluboké neuronové sítě (nákres na obr. 4.1). To jsou sítě, které se skládají z několika po sobě jdoucích skrytých vrstev. Obsahují tedy více něž tři vrstvy (včetně vstupní a výstupní). Každá vrstva se tak postupně může naučit vyšší a komplexnější úrovně abstrakce vstupu a vztahů.

4.1.1 Rekurentní sítě

Tato podkapitola vychází částečně z článku o LSTM sítích [23]. Vrstvy v klasických neuronových sítích či hlubokých neuronových sítích mají problém se zachycením časových posloupností akcí. V případě Starcraftu si takto můžeme představit prakticky celé hraní hry. Všechny akce mají určitou „setrvačnost“ a prodlevu, než se uskuteční jejich efekt. Např. pohyb jednotek není okamžitý, po kliknutí na příslušnou souřadnici na obrazovce trvá několik (desítek) snímků, než se dané jednotky na příslušné místo dostanou. Každá akce, tedy útočení, trénování jednotek, či stavění budov trvá dlouhé sekundy, což jsou v případě vnímaní agenta desítky a stovky snímků hry, které uplynou po zadání příkazu.



Obrázek 4.2: Architektura LSTM buňky. Předchozí skrytý stav $\boldsymbol{h}_{t-1}$ a vstup $\boldsymbol{x}_t$ jsou zpracovány pomocí tří bran $\boldsymbol{f}_t$, $\boldsymbol{i}_t$, $\boldsymbol{o}_t$, které regulují tok informací [39].

Zachytit tuto posloupnost je tak pro sítě prakticky nemožné. Buňky či vrstvy, které dokáží s tímto problémem bojovat, se nazývají rekurentní. V dnešní době se typicky používají například v oblasti rozpoznání řeči. Umožňují zachytit jakýsi stav přetrvání informace tím, že obsahují uvnitř „smyčky“, které předávají vstupní signál dále a disponují tak určitou formou „paměti“.

Mezi současné typicky používané rekurentní buňky patří tzv. LSTM buňky (angl. Long Short-Term Memory Cells), k vidění na nákresu 4.2. LSTM buňka obsahuje vnitřní stav $\boldsymbol{c}$, který je ovlivňován pomocí trojice bran $\boldsymbol{f}_t$, $\boldsymbol{i}_t$, $\boldsymbol{o}_t$:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \tag{4.1}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \tag{4.2}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \tag{4.3}$$

Kde $\boldsymbol{h}_{t-1}$ je předchozí skrytý stav, $\boldsymbol{x}_t$ je vstup, $\boldsymbol{b}$ je bias, σ je aktivační funkce *sigmoid* a $\boldsymbol{W}$ jsou váhy.

Brána $\boldsymbol{f}_t$ určuje množství informace, jaké buňka zapomene z předchozího stavu $\boldsymbol{c}_{t-1}$. Vstupní brána $\boldsymbol{i}_t$ určuje hodnoty, které budou aktualizovány a $\boldsymbol{o}_t$ určuje, jaké hodnoty stavu propustí na výstup. Kandidátní stav $\tilde{\boldsymbol{c}}_t$ buňky během výpočtu je definován jako:

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c) \tag{4.4}$$

Konečný vnitřní stav $\boldsymbol{c}_t$ a skrytý stav $\boldsymbol{h}_t$ se pak vypočítá následovně:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t \tag{4.5}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t) \tag{4.6}$$

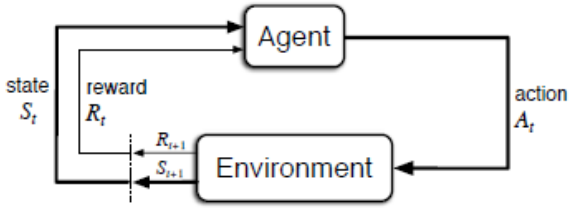
LSTM buňky jsou vylepšením „obyčejných“ rekurentních buněk a mohou si zapamatovat i delší sekvence kroků. Existuje i spousta dalších variant těchto buněk, např. LSTM buňky s kukátky, konvoluční LSTM nebo GRU buňky.

4.2 Posilované učení

Strojové učení se dá rozdělit na tři hlavní oblasti. Jedná se o učení s učitelem, učení bez učitele a posilované učení. V učení s učitelem jde typicky o problém klasifikace nějaké vstupní množiny dat. Podstata učení bez učitele je především v hledání jakési skryté struktury v datech. Posilované učení se od těchto dvou liší.

Posilované učení je učení toho, co udělat v nějakém prostředí. V tomto kontextu uvažujeme dvě entity, agenta a prostředí. Agent typicky interaguje s prostředím pomocí akcí, tím ho ovlivňuje a dostává od něj zpětnou vazbu. Agent k učení využívá signál od prostředí – odměnu, kterou může dostat po vykonání akce. Může obdržet buď kladnou odměnu v případě, že použil od něj očekávanou akci, nebo zápornou, v případě, že použil akci nevhodnou. Nebo může obdržet odměnu rovnu nule, pokud nenastal ani jeden z předchozích případů. Agent se tedy učí metodou „pokus-omyl“ a jeho cílem je danou odměnu maximalizovat.

Tento problém však není snadné řešit, jelikož odměny jsou obvykle zpožděné, tzn., že agent nemusí obdržet odměnu ihned po provedení akce, ale obdrží ji až za několik kroků, což stěžuje agentovo chápání, za jakou akci tuto odměnu vlastně dostal. Toto typicky nastává, pokud má daná akce nějaké dlouhodobější dopady. Druhý problém je, že odměny bývají řídké, nemusí tedy obdržet odměnu pokaždé, ale často třeba až na konci hry, např. +1 za výhru. Dalším problémem je dilema mezi zkoumáním a využíváním (angl. exploration vs.



Obrázek 4.3: Znázornění MDP. Agent v kroce t volí akci, obdrží od prostředí odměnu a přejde do nového stavu [32].

exploitation problem). Agent musí vybírat dobře ohodnocené akce, aby maximalizoval svoji odměnu, avšak musí také zkoumat prostředí a hledat potenciálně ještě lepší akce.

Formálně se problém posilovaného učení dá vyjádřit jako Markovský rozhodovací proces M (angl. Markov Decision Process), viz obr. 4.3, který je definován jako $M = (S, A, R, P)$, kde S je konečná množina stavů, A je konečná množina akcí, $R(s_t, a_t, s_{t+1})$ je funkce odměny a $P(r, s_{t+1}|s_t, a_t)$ je přechodová funkce, kde s je současný stav, a je zvolená akce ve stavu s_t, r je odměna po provedení akce a_t a s_{t+1} je nový stav [28].

Cílem agenta je maximalizovat celkovou sumu odměn, které získává v jednotlivých krocích t, maximalizuje tedy celkový očekávaný návrat (angl. expected return) G_t, který je definován [32]:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{4.7}$$

Kde T je koncový krok. Tento přístup se hodí typicky v epizodickém prostředí, což znamená, že interakce s prostředím je rozdělena na epizody, kdy po skončení epizody dojde k restartu prostředí a proces se opakuje bez návaznosti na předchozí.

Agent se může snažit maximalizovat zisk buď okamžitých odměn, nebo budoucích. Obvykle se však používá druhý zmíněný případ. Zavádí se proto parametr γ, takový, že platí $0 \leq \gamma \leq 1$, a který vyjadřuje váhu okamžitých odměn. Jedná se o redukční faktor (angl. discount faktor):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.8}$$

Pokud se faktor blíží k 0, je agent soustředěný na získávání okamžitých odměn. Naopak, pokud se blíží k 1, soustředí se více na budoucí odměny.

Agent se řídí pomocí strategie π, která je ve spojení s neuronovými sítěmi označována jako parametrizovaná stochastická strategie $\pi_\theta(a|s)$, kde θ jsou parametry (váhy) sítě a výstup strategie je rozdělení pravděpodobnosti pro akce a ve stavu s.

Pro ohodnocení stavů se používají dvě metriky. Funkce ohodnocení V (angl. state-value function) a funkce Q-ohodnocení Q (angl. state-action-value function) stavů s. Funkce ohodnocení stavu s, $V(s)$, je definována:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{4.9}$$

Pro všechna $s \in S$, kde $\mathbb{E}_\pi[\cdot]$ je očekávaná hodnota (angl. expected value), $V_\pi(s)$ je funkce ohodnocení stavu, pokud se řídíme strategií π, t je krok. Funkce ohodnocení říká,

jak „dobrý“ je daný stav, resp. jakou odměnu můžeme získat, pokud se budeme řídit strategií π. Q-funkce je definována následovně:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{4.10}$$

Kde $Q_\pi(s,a)$ je ohodnocení vybrání akce a ve stavu s, za předpokladu, že se řídíme strategií π. Q-funkce udává, jakou odměnu můžeme získat, pokud ve stavu s provedeme akci a, a poté se budeme řídit strategií π.

Kdybychom se řídili optimální strategií π^*, tak existují také optimální obě funkce ohodnocení:

$$V^*(s) = \max_\pi V_\pi(s) \tag{4.11}$$

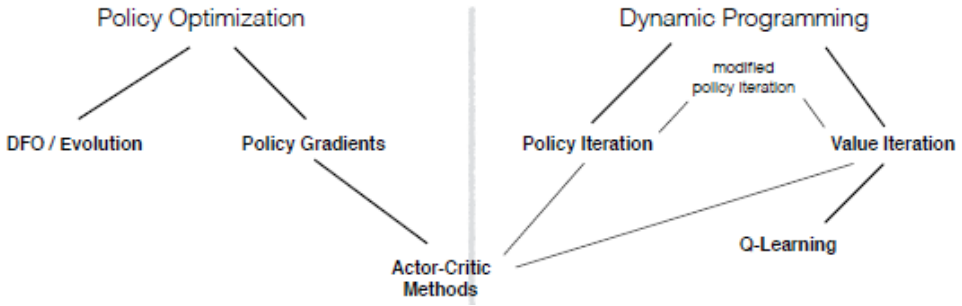
$$Q^*(s,a) = \max_\pi Q_\pi(s,a) \tag{4.12}$$

Pro všechna $s \in S$, $a \in A$. Funkce Q^* se dá taktéž vyjádřit pomocí V^*:

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma V^*(s_{t+1}) \mid S_t = s, A_t = a] \tag{4.13}$$

4.2.1 Hluboké posilované učení

Mezi moderní metody posilovaného učení patří tzv. hluboké posilované učení (angl. Deep reinforcement learning). Termín „hluboké“ se váže k použití hlubokých neuronových sítí využívaných jako aproximátory funkcí. Samotná myšlenka hlubokého posilovaného učení nová není, již v 90. letech existovaly programy, které ho využívaly, a byly díky němu schopny hrát např. vrhcáby [34]. Nicméně největší rozmach zažilo v poslední době po úspěchu v oblasti zpracování obrazu a řečových signálů a také po demonstraci schopnosti hrát téměř padesátku různých her na Atari [21].



Obrázek 4.4: Rozdělení metod řešení posilovaného učení. Metody se dělí na tři hlavní části, optimalizace strategie, dynamické progamování a aktér-kritici [28].

Řešení problémů posilovaného učení probíhá typicky pomocí tří metod, viz obr. 4.4. Jedná se o optimalizaci strategie, dynamické programování nebo hybridní metody aktér-kritiků. Metody optimalizace strategie se zaměřují na strategii, snaží se optimalizovat očekáváné odměny s ohledem na strategii. Sem spadají např. evoluční algoritmy nebo PG metody (angl. policy gradient). PG metody slouží k postupnému a opakovanému upravování gradientů strategie. Obvykle se k tomu používají optimalizační techniky typu SGD

(angl. stochastic gradient descent). Přístup dynamického programování se zaměřuje spíše na naučení funkce ohodnocení, které udává, jak vysokou odměnu agent může získat. K těmto metodám se řadí např. iterování strategie nebo iterování funkce ohodnocení. Třetí přístup, aktér-kritici, je hybridní postup mezi oběma zmíněnými. Snaží se optimalizovat strategii, ale zároveň k tomu využívá funkci ohodnocení.

Metody založené na optimalizaci strategie typicky aproximují parametry strategie pomocí optimalizačních technik gradientu. Jedny z těchto metod jsou algoritmy *REINFORCE*. Ty upravují parametry strategie ve směru [20]:

$$\nabla_\theta \log \pi(a_t|s_t;\theta)(R_t - b_t(s_t)) \tag{4.14}$$

Kde výraz $(R_t - b_t(s_t))$ redukuje varianci, přičemž pro $b_t(s_t)$ se typicky používá naučený odhad funkce ohodnocení, tedy $b_t(s_t) \approx V^*(s_t)$. Zmíněný výraz je možné pochopit jako funkci prospěchu A (angl. Advantage function), která vyjadřuje míru (pozitivní nebo negativní) ohodnocení stavu, než jaká byla předpokládaná. Pokud bylo ohodnocení vyšší, než které bylo očekáváno, je funkce A kladná, a naopak, pokud bylo nižší, je funkce záporná. A je tedy definována jako:

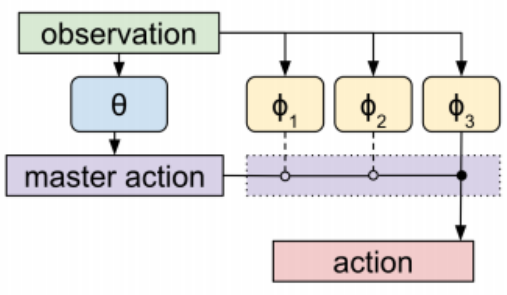
$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t) \tag{4.15}$$

Jelikož R_t je odhad $Q^\pi(a_t, s_t)$ a b_t je odhad $V^\pi(s_t)$, což je základ aktér-kritik metod [20].

4.2.2 Hierarchické posilované učení

Hierarchie učení představuje abstrakci pro vykonávání nízkoúrovňových akcí s cílem splnit nějaký vyšší cíl. Takové chování je běžné i v lidském životě. Vykonání každé akce se skládá z posloupnosti či kombinace různých dílčích akcí. Lidé si tyto činnosti prakticky již ani neuvědomují a provádějí je podvědomě. Toto chování je možné přirovnat k hierarchii chování agenta, kdy agent zvolí nějakou vysokoúrovňovou akci, kterou chce provést a tato akce je nízkoúrovňově vykonávána dílčími agenty.

První zmínky o hierarchickém chování agentů pocházejí od R. Suttona, který se o nich zmiňuje jako o volbách (angl. options). Popisuje je jako sub-strategie vykonávané po určitou dobu činnosti. Jsou formálně definovány jako semi Markovské rozhodovací procesy [33].



Obrázek 4.5: Hierarchická architektura agenta. Agent hlavní strategií vybírá akce, které jsou abstrakcí vykonávání jednotlivých sub-strategií [12].

V případě hierarchie strategií mohou být jednotlivé sub-strategie předem natrénovány na dílčích úkolech a poté vykonávány [25]. Druhou možností je, že sub-strategie nejsou

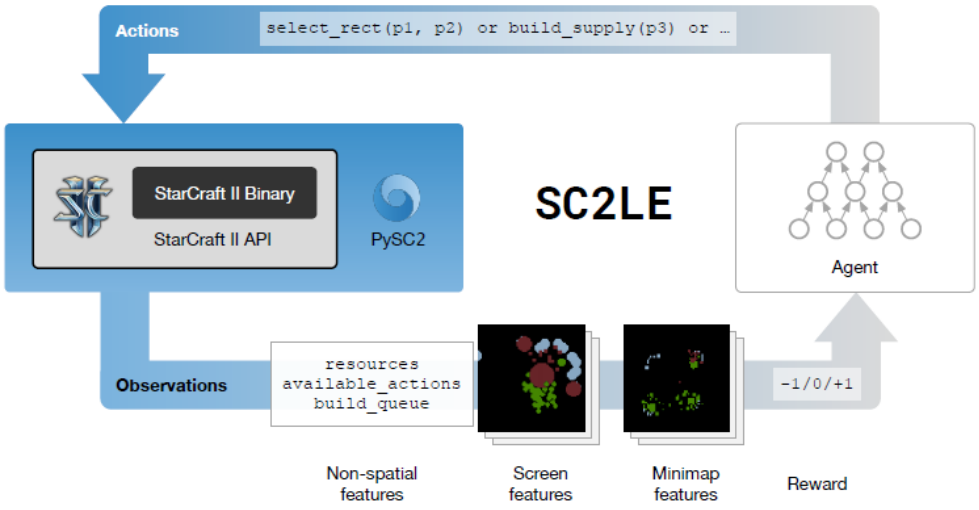
pevně dané a jsou naučeny pomocí meta učení [33]. Pomocí meta učení je možné využít natrénované znalosti efektivně na nových, doposud neviděných problémech. Výhoda těchto metod spočívá v tom, že je možné je kombinovat s metodami řešení hlubokého posilovaného učení, tedy třeba pomocí PG metod nebo aktér-kritiků.

Pomocí hierarchických a meta metod je možné řešit různé velmi složité úkoly bez toho, aniž by agent disponoval nějakou expertní znalostí problému předem. Vyobrazení zmíněné hierarchie je na diagramu 4.5.

4.3 PySC2

DeepMind ve své práci [35] představil nové učící prostředí pro Starcraft II, souhrnně nazývané SC2LE (angl. Starcraft II Learning Environment), ve kterém mohou být snadno testovány a trénovány algoritmy založené na bázi strojového učení pro hraní hry Starcraft II. Doposud žádné takové prostředí nebylo vytvořeno. DeepMind tak ve spolupráci s Blizzardem učinil velký krok kupředu k řešení problematiky umělé inteligence ve Starcraftu, ale případně i v jiných strategických i nestrategických hrách.

SC2LE se skládá ze tří hlavních částí, těmi jsou: binární kód hry, API pro komunikaci s herním enginem a PySC2 (Python Starcraft 2). PySC2 je volně šiřitelné prostředí naprogramované v Pythonu, které komunikuje se zpřístupněným API Starcraftu, viz diagram 4.6.

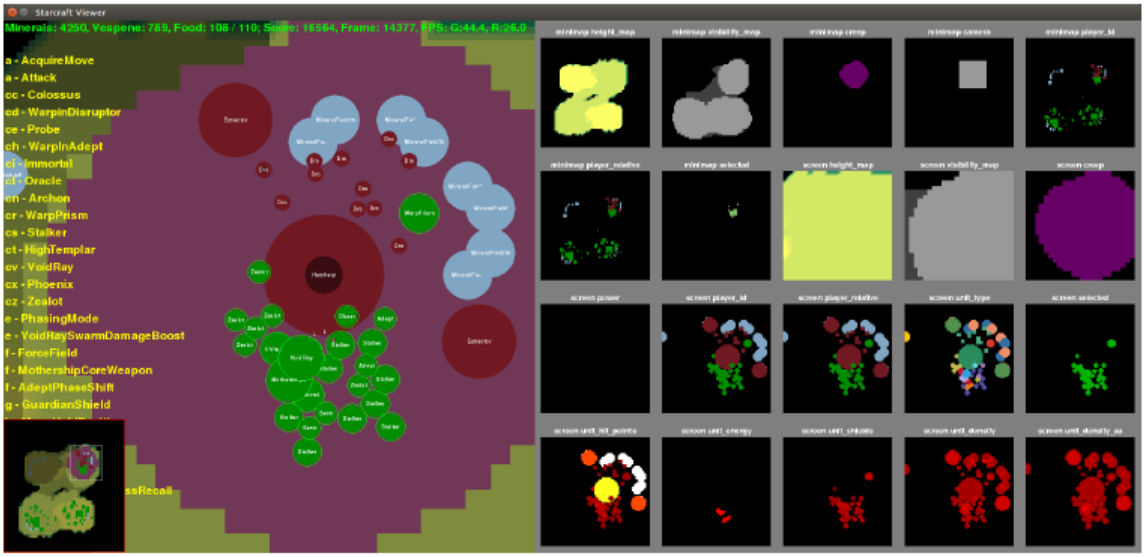


Obrázek 4.6: Diagram SC2LE. PySC2 poskytuje agentovi informace ohledně hry skrze pohledy, které obsahují dostupné akce, vrstvy rysů pro obrazovku a minimapu i vektor statistik [35].

PySC2 nabízí abstrakci modelu prostředí Starcraftu pomocí vrstev rysů (angl. Feature Layers), které zprostředkovávají agentovi pohled na svět. Agent tedy prozatím nemá možnost rozpoznávat stav hry přímo z pixelů. Dále je v každém kroku k dispozici seznam dostupných akcí, které může agent zvolit a vektor statistik, který obsahuje všechny ostatní dostupné informace (např. množství natěžených surovin, počet jednotek apod.).

Vrstvy rysů jsou rozděleny do dvou skupin, rysy pro obrazovku (tj. momentálně vybraná část celé hrací plochy náhledovou kamerou), kterou hráč momentálně vidí (Screen Feature Layers), a rysy pro minimapu (celková reprezentace hrací plochy, která neobsahuje tolik

detailních informací jako obrazovka, ale poskytuje pohled na hru jako celek), která se váže k právě hrané hře (Minimap Feature Layers). Pro obě sady vrstev je k dispozici několik typů informací o hře, např. vrstvení terénu, viditelnost mapy, typy jednotek na mapě, množství zbývajících životů jednotek apod. Grafickou reprezentaci vrstev je možné vidět na obr. 4.7.



Obrázek 4.7: Ukázka vrstev rysů v PySC2. Na pravé straně je možné vidět všechny dostupné vrstvy rysů, např. odkrytá část minimapy, typy jednotek, terén apod. Vlevo jsou pak vrstvy rysů interpretované pro člověka, který může skrze ně také hrát hru [35].

PySC2 dále také nabízí další nezbytné prvky pro agenta, aby mohl hrát hru. Tím je seznam proveditelných akcí v každém stavu hry, vektor statistik, který obsahuje všechny ostatní dostupné informace (např. množství natěžených surovin, počet jednotek apod.) a signál o získání odměny, který je pro agenta esenciální. Odměny mohou být řídké, ve formě $+1/0/-1$ za celý zápas za výhru/remízu/prohru, anebo pomocí skóre, které je zabudované ve hře a typicky není dostupné hráčům během hry. Jedná se o kumulované součty za všechny postavené jednotky, budovy, vylepšení, natěžené suroviny, zabité nepřátelské jednotky apod., a mínusové skóre za každou zabitou vlastní jednotku nebo zničenou budovu. Toto ohodnocení tak může dosahovat velkých (desetitisíce nebo klidně i statisíce) hodnot během hry.

Vývoj umělé inteligence pro Starcraft II se zaměřuje prozatím pouze na 1v1 mód, tedy souboje mezi dvěma protivníky. Tento mód je také ze všech nejpopulárnější i v profesionální soutěžní lize. Hru je možné hrát také v jiných módech na týmy, např. 2v2, 3v3 nebo 4v4, což přináší již velmi komplexní hry a také případné nevyváženosti v týmech (např. 4v2).

Samotná hra již obsahuje umělou inteligenci v různých úrovních obtížnosti, od nejlehčí, až po velmi těžkou. K dispozici je ještě vyšší úroveň, ale ta je zvýhodněná oproti hráči, jelikož podvádí, resp. těží rychleji suroviny, má lepší vizi na mapě apod. Každá úroveň obtížnosti bota je však skriptovaná, tedy bot vybírá ze sady předem definovaných pravidel a taktik.

4.4 Tensorflow

Tensorflow[9] je otevřený software, v současnosti využívaný především pro programování modelů strojového učení, zejména neuronových sítí. Tensorflow byl vyvinut společností Google Brain a v současnosti je jí hojně využíván. Google pro svůj kód psaný v Tensorflow vyvinul i unikátní integrované čipy (TPU[10]), které značně urychlují tenzorové výpočty. Existují i specializované nadstavby nad Tensorflow, např. Sonnet[11] od DeepMind [1].

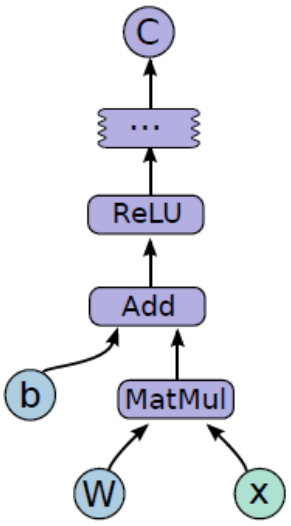
Základem Tensorflow je výpočetní graf. Jedná se o orientovaný graf, který se skládá z uzlů. Graf je pak abstraktním zobrazením toku dat během výpočtu a je typicky sestaven při prvotním spuštění programu a nadále zůstává neměnný. Tensorflow podporuje několik programovacích jazyků, např. Python, C++, nebo Javu. Příklad výpočetního grafu je na obr. 4.8. Každý uzel má N vstupů a M výstupů, kde $N, M \geq 0$, a reprezentuje jednu operaci. Uzly jsou spojeny hranami, přičemž hrana představuje počítané hodnoty, které jsou reprezentovány pomocí tenzorů. Operacemi jsou v Tensorflow myšleny veškeré příkazy, které je možné využít, např. násobení matic (`tf.matmul`), sčítání dvou hodnot (`tf.add`) apod.

Dalšími nezbytnými prvky, se kterými Tensorflow pracuje jsou proměnné a zástupci. Proměnné přetrvávají po celou dobu běhu programu a slouží k uchování hodnot, které je možné upravovat (trénovat), např. váhy neuronových sítí. Zástupci jsou pak zástupné symboly pro operace, jejichž hodnota je naplněna až za běhu programu.

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))
W = tf.Variable(tf.random_uniform([784,100],-1,1))
x = tf.placeholder(name="x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]


s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...
  result = s.run(C, feed_dict={x: input})
  print step, result
```



Obrázek 4.8: Ukázka výpočetního grafu v Tensorflow. Vlevo je kód zapsaný v Pythonu pomocí Tensorflow a vpravo je výpočetní graf, který odpovídá danému kódu [1].

S již sestaveným grafem se pak za běhu programu pracuje pomocí sezení. Skrze sezení pak lze zavolat metodu `run`, která vypočítá požadovanou operaci. `Run` vyhodnotí pouze uzly, které jsou nezbytné pro výpočet zadané operace, případně další explicitně specifikované. Kromě požadovaného seznamu operací, který je předán metodě `run` pomocí argumentů, je

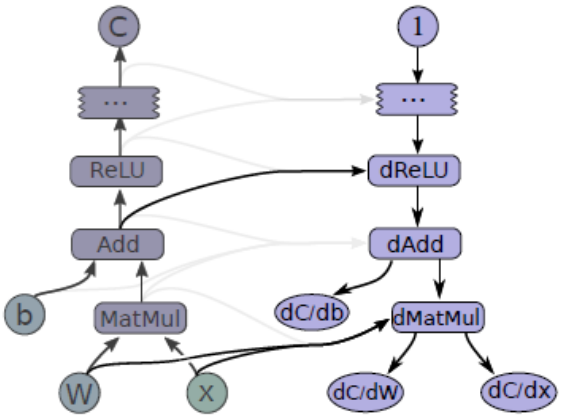
[9] www.tensorflow.org

[10] TPU (Tensor Processing Unit) je integrovaný obvod vyrobený speciálně pro umělou inteligenci. Tyto čipy mohou být několikrát rychlejší než současná GPU a CPU [15].

[11] https://github.com/deepmind/sonnet

taktéž třeba předat argument `feed_dict`, který obsahuje všechny nezbytné hodnoty pro výpočet operací, kterými se naplní zástupci. Právě pomocí opakovaného volání `run` probíhá obvykle vyhodnocování grafu a počítání hodnot.

Tensorflow dále umožňuje automatický výpočet gradientu (viz obr. 4.9), což se především využívá pro různé optimalizační techniky k hledání minima funkcí, např. v SGD.



Obrázek 4.9: Znázornění výpočtů gradientu v Tensorflow. Vlevo je vidět výpočetní graf operací, vpravo pak derivace jednotlivých operací, přičemž černé šipky zobrazují využité vstupy pro výpočet gradientů a šedé šipky potenciální vstupy pro výpočet, které nejsou aktuálně potřeba [1].

Jelikož se na dnešních systémech vyskytuje větší počet zařízení (procesory a grafické karty), je Tensorflow k dispozici ve dvou verzích. První, označována jako *tensorflow*, slouží pro vykonávání kódu čistě na jádrech procesorů. Druhá verze, *tensorflow-gpu*, umožňuje využít i grafické karty pro výpočet, což ve většině případů značně urychlí výpočty, a tím pádem i učení neuronových sítí. Tensorflow taktéž podporuje běh programu jak na jednom fyzickém stroji, tak na mnoha strojích pomocí distribuování operací mezi jednotlivé clustery.

Kromě samotné knihovny pro výpočty je k dispozici i vizualizační nástroj Tensorboard, pomocí kterého lze snadno vizualizovat vytvořené modely. Vzhledem k tomu, že mnohé modely sítí jsou velmi složité a velké, vizualizace ulehčuje vývojářům práci díky jednoduchému a interaktivnímu zobrazení jejich modelů. Pro lepší přehlednost mohou být jednotlivé uzly, resp. operace, pojmenovány pomocí argumentů `name` a `scope`. Dále je možné vytvářet histogramy a uchovávat prakticky jakékoliv skalární hodnoty, obrázky nebo audio.

Kapitola 5

Implementace

Tato kapitola se zabývá implementací jednotlivých částí bota. Vytvořený program staví na práci týmu DeepMind a snaží se hrát hru jako celek, nikoliv pouze některé její aspekty (např. se zaměřením pouze na souboje).

Všechny implementované programy byly psány v jazyce Python ve verzi 3. Volba programu byla logická, jelikož je prostředí PySC2 psané také v jazyce Python a lze s ním tak snadno komunikovat. Python je celkově v oblasti strojového učení velmi populární jazyk pro implementaci různých modelů. Lze předpokládat, že je to díky tomu, že syntax kódu je relativně snadno pochopitelná, jazyk je to všestranný, efektivní při zpracování velkého množství dat a při matematických operacích s maticemi, což je prakticky základní stavební kámen neuronových sítí. Další nespornou výhodou je existence velkého množství modulů a balíků, které obsahují již implementované nejrůznější algoritmy. K nejznámějším balíkům patří např.: numpy, scikit, pandas a spousta dalších. Mezi nevýhody pak může patřit především pomalejší výkon a horší efektivita při paralelním zpracování dat a výpočtů. K práci se strojovým učením pak byla využita knihovna Tensorflow.

Bot využívá PySC2 ve verzi 1.2, sc2clientprotocol 4.1.1 a klienta Starcraftu 2 (pro Linux) 4.0.2. Z dalších hlavních balíků je pak používán Tensorflow ve verzi 1.4.1 a numpy 1.13.3.

Kapitola je členěna na několik sekcí. Nejprve budou podrobně popsány použité a navržené modely hlubokých neuronových sítí, které tvoří jádro celého bota. Poté budou jednotlivě popsány implementace učení modelu ze záznamů z her hráčů jakožto typ učení s učitelem a posléze bude rozebráno trénování agenta pomocí technik posilovaného učení. Dále budou uvedena omezení a redukce, kterými je program limitován a které umožňují snížit komplexnost hry a modelu jako takového. Pokud by byl uvažován plný potenciál hry, byla by výpočetní náročnost modelu velmi vysoká.

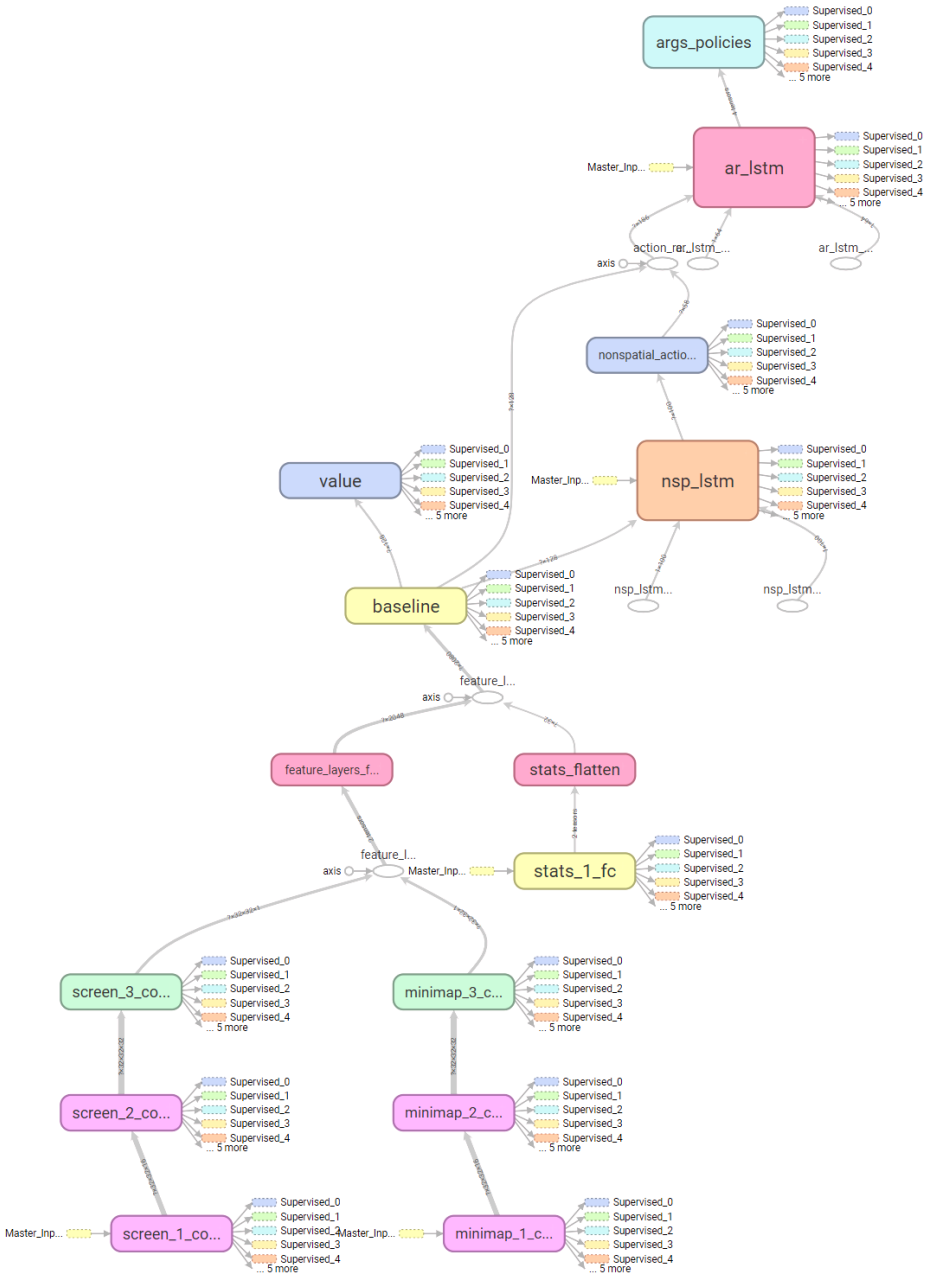
5.1 Model neuronových sítí

Návrh sítí vychází částečně ze sítí, které ve své práci použil DeepMind. Označuje je jako *Atari-net* a *FullyConv*, jedná se o hluboké neuronové sítě, přičemž konkrétní popis je možné najít v jejich práci [35].

Navržený model v této práci využívá celkem čtyři rozdílné hluboké neuronové sítě, přičemž každá řeší jiný úkol. Tři sítě využívají stejnou architekturu a jedna odlišnou. Koncept je takový, že jedna hlavní síť řídí zbylé tři. Konkrétně se jedná o řídící síť N_A, nazvanou jako *ArbiterNetwork*, která se stará o přepínání mezi ostatními. Zbylá trojice pak tvoří podskupinu, resp. podsítě, kde každá řeší odlišnou problematiku hry.

Jedná o sítě *BuilderNetwork* N_B, která se stará o stavění budov a přiřazování práce těžebním jednotkám, dále je to *TrainerNetwork* N_T, která řeší trénování vojenských a těžebních jednotek a *CombatNetwork* N_C, která řídí vojenské jednotky, umožňuje jim používat jejich speciální schopnosti a útočí na nepřítele. Právě tato trojice sítí N_B, N_T, N_C sdílí stejnou architekturu, která je uvedena na diagramu 5.1. Síť N_A pak používá architekturu mírně odlišnou, viz diagram 5.2.

Cílem je vytvořit model, který bude schopen řešit komplexní hraní Starcraftu tak, že jej rozdělí na menší podproblémy a tím tak přiblížit hraní jakési hierarchii.



Obrázek 5.1: Architektura herních sítí.

5.1.1 Architektura herní sítě

Základ tvoří dvě konvoluční neuronové sítě skládající se ze tří vrstev, které zpracovávají vstupní vrstvy rysů. První vrstva obsahuje 16 filtrů o velikosti 5x5. Druhá 32 filtrů o velikosti 3x3. Obě vrstvy zachovávají rozlišení vstupních snímků a používají proto krok o velikosti 1 a zarovnání. Třetí vrstva je tvořena lineárním 1x1 filtrem. Tyto konvoluční části zpracovávají odděleně vstup pro vrstvy rysů obrazovky a vrstvy rysů minimapy. Na diagramu 5.1 tomuto odpovídají vstupní bloky označené jako *screen* a *minimap conv2d*.

Třetím vstupem do sítě je vektor statistik, který je přiveden na vstup plně propojené vrstvy *stats_1_fc* s 32 neurony. Výstup z obou konvolučních sítí a plně propojené vrstvy je dále sloučen a reprezentuje tak celkový vstupní stav. Na ten navazuje plně propojená vrstva *baseline* se 128 neurony. Z této vrstvy vedou tři výstupy, prvním je plně propojená vrstva s 1 neuronem *value*, která je zároveň jedním z výstupů celé sítě a vyjadřuje funkci ohodnocení V. Těmi dalšími jsou dvě LSTM buňky *nsp_lstm* se 100 neurony a *ar_lstm* se 64 neurony. Na první zmiňovanou navazuje plně propojená vrstva *nonspatial_actions_policy* s 58 neurony, která reprezentuje akce, které může agent provést, a je zároveň dalším výstupem sítě a také vstupem pro *ar_lstm*. Z této LSTM buňky pak vedou čtyři plně propojené vrstvy, každá o 32 neuronech, dohromady označené blokem *args_policies* a reprezentují možné prostorové argumenty akcí.

Parametr *dropout* byl nastaven během trénování na 40 %. Pro lepší přehlednost slouží diagram 5.1, se kterým korespondují použité názvy vrstev. Výstupů sítě je tedy celkem šest, funkce ohodnocení, akce a čtveřice možných argumentů.

5.1.2 Architektura řídící sítě

Design řídící sítě je ze začátku totožný jako v případě herní sítě, tedy výše uvedenou architekturu je možné sledovat až do bodu vytvoření sloučeného stavu ze vstupních vrstev rysů a statistik, viz diagram 5.2. Po sloučení vrstev se však síť mírně liší. Následuje plně propojená vrstva *baseline* s 32 neurony. Z ní pak vedou dva výstupy – jedna plně propojená vrstva s 1 neuronem *value* a druhá LSTM buňka *arbiter_lstm*. Ta se skládá ze 100 neuronů. Od ní pokračuje plně propojená vrstva se třemi neurony *action_policy*, která reprezentuje akce, které může síť provést. Tyto akce značí dané podsítě, které budou aktivní. Parametr *dropout* byl během trénování nastaven taktéž na 40 %. Výstup sítě je tvořen funkcí ohodnocení a akcemi, které může daná síť zvolit.

5.1.3 Aktivační funkce

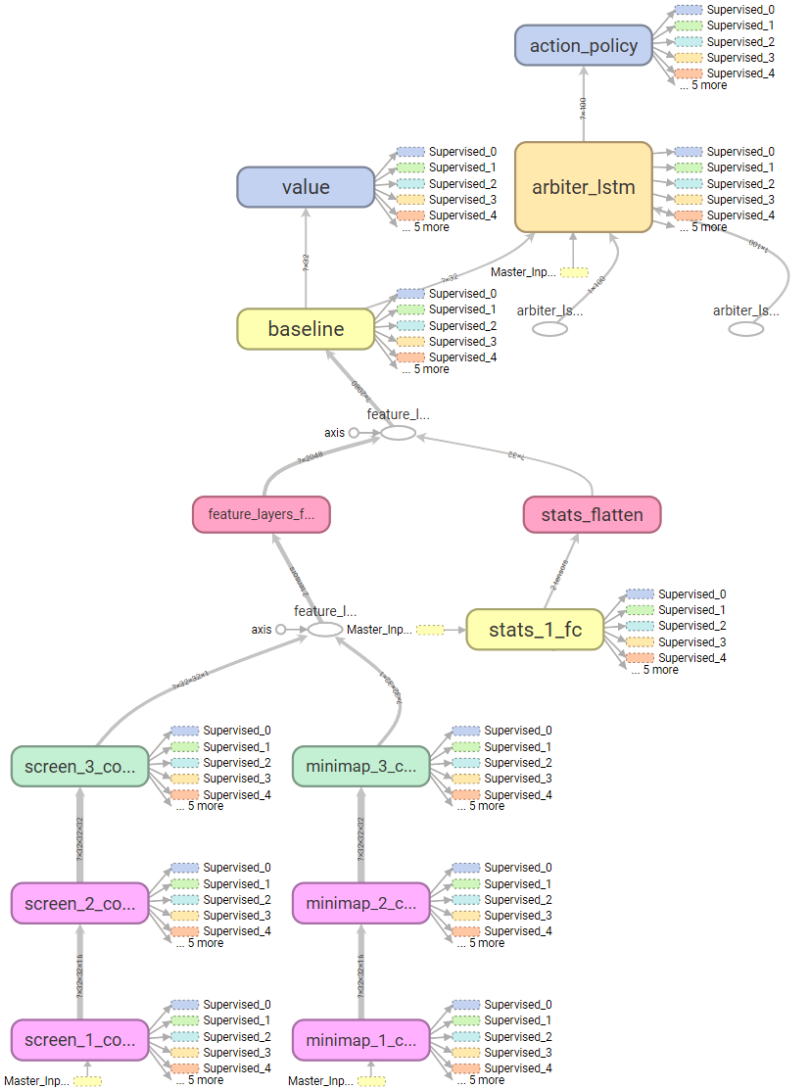
Všechny výstupy neuronových sítí jsou následně zpracovány pomocí funkce *softmax*. *Softmax* je logistická funkce, která se dá použít pro výpočet rozdělení pravděpodobnosti akcí ze vstupních hodnot. Výstup tvoří reálná čísla v intervalu $[0; 1]$, přičemž jejich součet je roven 1. *Softmax* je definován následovně [14]:

$$softmax(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} exp(x_j)} \tag{5.1}$$

Kde $\boldsymbol{x}$ je vektor vstupních hodnot, i je aktuální hodnota.

Ostatní vrstvy sítě využívají usměrňovací aktivační funkce. Typickým příkladem takové nejjednodušší funkce je *ReLU* (angl. Rectified linear unit). Funkce *ReLU* je definována následovně [13]:

$$f(x) = max(x, 0) \tag{5.2}$$



Obrázek 5.2: Architektura řídící sítě.

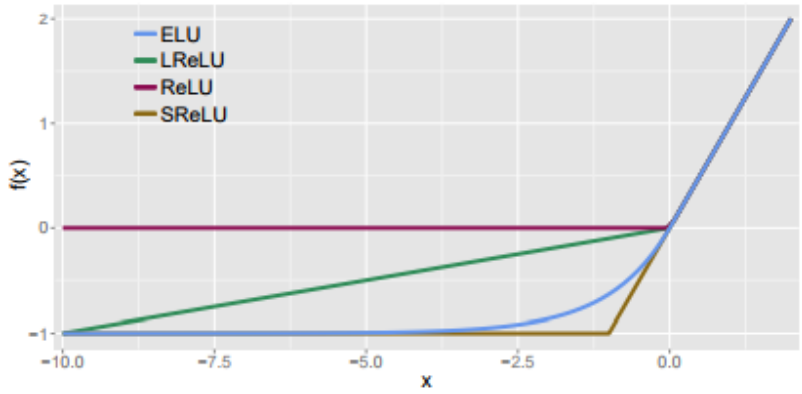
ReLU však v praxi může trpět problémem mizejících gradientů (angl. vanishing gradients), což znamená že velká hodnota gradientu může ovlivnit váhu neuronu natolik, že se jeho hodnota bude rovnat 0 a nadále tuto hodnotu již není možné změnit, čili tento neuron „odumře“. V praxi se tak může dít až u několika desítek procent neuronů v sítí. Tomuto jevu se dá určitými technikami zabránit (např. ořezáním hodnot gradientů, snížení koeficientu učení apod.) a ačkoliv DeepMind pro svoji síť *ReLU* využívá, v této práci se osvědčilo využít modifikaci této aktivační funkce, a tou je *ELU* (angl. Exponential Linear Unit). Její výhoda spočívá především v tom, že neurony nemohou „odumřít“. Výpočet je ovšem o něco více náročnější. *ELU* je definována následovně [11]:

$$f(x) = \begin{cases} x & \text{pokud } x > 0 \\ \alpha \cdot (e^x - 1) & \text{pokud } x \leq 0 \end{cases} \tag{5.3}$$

Kde α je hyperparametr, přičemž $0 \leq \alpha$.

Usměrňovacích funkcí je celá řada, je možné najít jejich různé obdoby, např. *LReLU* (angl. Leaky ReLU) nebo *PReLU* (angl. Parametric ReLU), které taktéž netrpí problémem mizícího gradientu. Srovnání průběhu některých aktivačních funkcí je na obr. 5.3.

Výjimku tvoří plně propojené vrstvy *value*, která používají lineární aktivaci. Tu využívají i výstupy sítě pro akce a jejich argumenty, které navazují na LSTM buňky.



Obrázek 5.3: Srovnání průběhu některých usměrňovacích aktivačních funkcí. Na grafu je kromě výše zmíněných *ReLU*, *ELU* a *LReLU* také *SReLU* (angl. Shifted ReLU) [11].

5.2 Omezení

Pro zjednodušení byl bot naprogramován tak, aby mohl hrát pouze za jednu rasu, a to Terrany. Tato rasa byla zvolena vzhledem k jejím ne až tak složitým mechanikám, na rozdíl od jiných. Např. Zergové vyžadují stavění budov na *creepu*, což je označení pro speciálně modifikovaný terén mapy, na kterém mají povoleno stavět, a který se pozvolna rozšiřuje od jejich základny. Také kvůli některým dalším omezením, např. rozlišení, by nebylo možné za Zergy reprezentovat některé jednotky, jelikož by byly již příliš malé pro rozpoznání z vrstev rysů. Program se dále zaměřuje na 1v1 mód zápasů proti rase Terran.

5.2.1 Redukce dimenzí

V praxi bylo použito několik úprav pro snížení výpočetní náročnosti modelu. Rozlišení vrstev rysů obrazovky i minimapy bylo sníženo na 32x32 pixelů. Bylo otestováno, že toto rozlišení je ještě dostatečné pro člověka i bota, aby byl schopen rozeznat všechny objekty na obrazovce, a téměř všechny objekty na minimapě. Některé objekty, např. mnoho nahromaděných jednotek na jednom místě, nejsou úplně korektně reprezentovány na minimapě při tomto rozlišení, ale dané nevýhody jsou zanedbatelné. Díky tomuto omezení se výrazně sníží množství parametrů a tím i výpočetní náročnost a paměťové nároky modelu. Ve hře existují i jednotky, které by při tomto rozlišení nebylo možné identifikovat na obrazovce, nicméně jedná se pouze o jednotky za rasu Zerg a můžeme si tedy dovolit tento fakt ignorovat.

Celkový možný počet vrstev rysů, které podporuje PySC2, udávající rozdílné charakteristiky je 24 (z toho 17 pro obrazovku a 7 pro minimapu). Při využití všech rysů by pak vstup do sítě tvořily snímky obrazovky o velikosti 32x32x17 (17408 hodnot), a snímky minimapy o velikosti 32x32x7 (7168 hodnot), tj. 24576 hodnot. V případě ponechání rozlišení 64x64 by to bylo celkově 98304 hodnot. Z těchto rysů jsou vybrány pouze ty základní a dostačující pro plné hraní hry. Jedná se o následující rysy pro obrazovku:

- *visibility_map* – zobrazuje viditelnost na právě se nacházející obrazovce, tedy jaká část je skryta a jaká odkryta.
- *player_relative* – zobrazuje jednotky a budovy dle jejich vztahu k hráči (vlastní jednotky, nepřátelské, neutrální).
- *unit_type* – zobrazuje typy (identifikátory) jednotek či budov na obrazovce, aby je bylo možné typově rozlišit.
- *selected* – zobrazuje aktuálně vybrané jednotky, popř. budovy, na obrazovce, kterým je možné dávat příkazy.
- *unit_hit_points* – zobrazuje zbývající životy jednotek a budov na obrazovce.

A rysy pro minimapu:

- *visibility_map* – zobrazuje viditelnost na celé minimapě, tzn. jaká část minimapy je odkrytá.
- *camera* – zobrazuje aktuální výřez minimapy, na kterém je kamera obrazovky.
- *player_relative* – zobrazuje jednotky a budovy dle jejich vztahu k hráči na minimapě.

Kompletní přehled možných vrstev je dostupný v oficiální dokumentaci[12]. Celkově je tedy využito pouze 8 vrstev rysů, 5 pro obrazovku a 3 pro minimapu, které postačují pro dostatečné hraní člověku i automatickému systému.

5.2.2 Akce

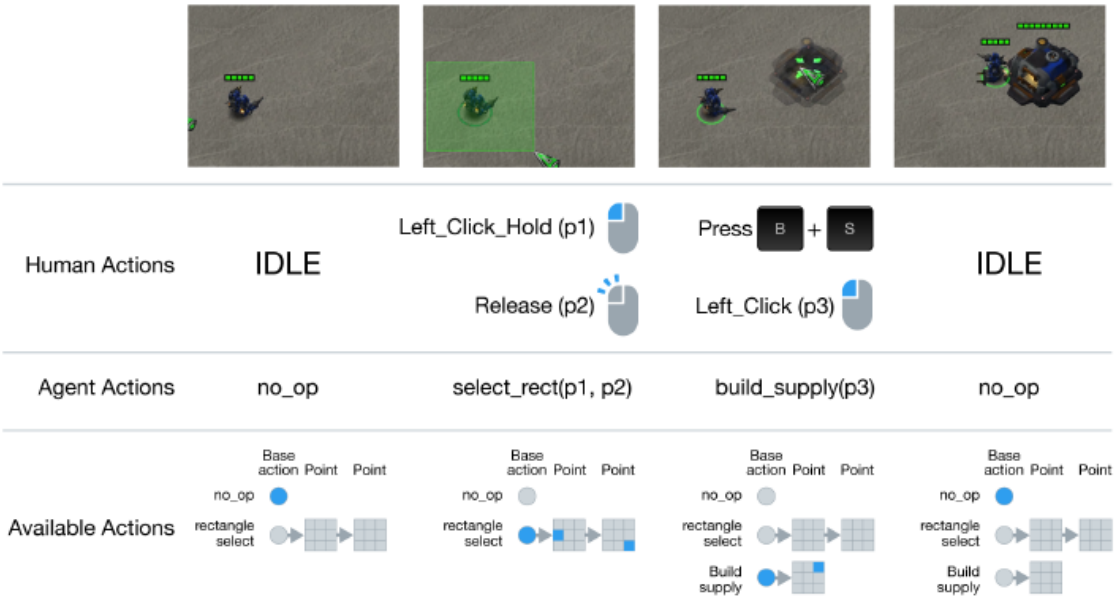
Ve hře je k dispozici teoreticky až 524 různých akcí (bez uvažování prostorových akcí jakožto souřadnic), které agent může zvolit. Některé z nich se dají zařadit pod své generické typy, a tudíž z tohoto počtu odečíst. Jelikož skrze PySC2 je stejně možné přistoupit pouze k těm generickým. Pokud budeme uvažovat pouze zvolenou rasu Terran a ponecháme pouze generické akce, které může tato rasa použít, jedná se o asi 150 rozdílných akcí, tzn. jsou zde zahrnuty kromě základních, např. trénování všech dostupných jednotek, budov, všechna vylepšení pro danou rasu, i všechny speciální útoky, kterými jednotky mohou disponovat.

Z tohoto počtu bylo využito pro trénování na záznamech celkem 58 akcí, které se dají rozdělit na zhruba čtyři skupiny. Jedná se o stavění budov (14 akcí), trénování jednotek (15 akcí), výzkum (10 akcí) a ostatní (19 akcí). Pro stavění budov, resp. trénování jednotek, byly využity všechny dostupné akce pro rasu Terran. Z akcí zkoumajících vylepšení bylo zvoleno 10 akcí, které mohou mít potenciálně výraznější vliv na hraní. Mezi ostatní akce se pak řadí především obecné akce typu posun kamery, označování jednotek a speciální schopnosti jednotek. Rozdělení akcí do skupin je k náhledu na obr. 6.4 anebo podrobně pro jednotlivé sítě v příloze B.

Každá akce může mít několik různých argumentů. Jednotlivé argumenty pro akce jsou reprezentovány pomocí řetízkového pravidla, jenž je definováno následovně [35]:

$$\pi(a \mid s) = \prod_{l=0}^{L} \pi(a^l \mid a^{\leq l}, s) \tag{5.4}$$

[12] https://github.com/deepmind/pysc2/blob/master/docs/environment.md



Obrázek 5.4: Srovnání vykonávání akcí ve hře. Cílem akce je označit jednotku a zadat ji příkaz stavby budovy. První řádek ukazuje akci z pohledu člověka, ten musí vybrat jednotku myší, poté zvolit budovu z nabídky a na závěr kliknout na místo, kde ji chce postavit. Druhý a třetí řádek ukazují akci z pohledu agenta a jak ji prostředí PySC2 reprezentuje. PySC2 umožňuje vykonání jednotlivých akcí, až pokud pro ně byly splněny podmínky, tedy pro postavení budovy musí agent nejprve danou jednotku označit, čímž přibližuje hraní agenta k hraní skutečného člověka [35].

Kde s je stav a a^l je argument dané akce. Toto pravidlo umožňuje transformovat rozlehlý prostor možných akcí do posloupnosti L akcí. Znázornění provádění akcí je na obr. 5.4.

Například akce *select_rect*, která simuluje výběr označením pomocí myši na obrazovce, vyžaduje tři argumenty. Těmi jsou *select_add*, *screen*, a *screen2*. První argument udává, zda se mají jednotky přidat do předchozího výběru, zbylé dva jsou pak prostorové souřadnice na mapě pro výběr. Naproti tomu třeba akce *noop* (nedělej nic) nevyžaduje žádný argument.

5.3 Učení ze záznamů

Před samotným učením ze záznamů bylo třeba vytvořit program, který dané záznamy zpracuje, tzn. převede z interního formátu Blizzardu do nějakého čitelného. K tomuto účelu byl vytvořen program *PyReplayParser*, taktéž psaný v Pythonu, který s pomocí PySC2 převede originální záznamy do *pickle* souborů.

PyReplayParser je možné spustit s přepínačem `--threads` a `--mode`, kde prvně zmiňovaný udává v kolika vláknech budou zpracovávány záznamy a druhý určuje, v jakém módu se má program spustit, tedy, zdali má extrahovat metadata (hodnota přepínače `info`), přesunout vybrané soubory (hodnota `move`), či zahájit zpracování záznamů (hodnota `parse`). Program předpokládá, že se všechny záznamy budou nacházet ve složce *Replays* v kořenovém adresáři hry. Respektive je nutné nastavit nějakou cestu pro PySC2 pomocí proměnné `SC2PATH`. Byla tedy zvolena výchozí složka, kterou je zmíněný kořenový adresář.

V následující podkapitolách bude tento program podrobněji popsán a rozebrána konkrétní implementace jednotlivých částí.

5.3.1 Extrakce metadat

Před zpracováním surového záznamu je nejprve nutné extrahovat metadata, která obsahuje, abychom věděli, zda je daný záznam vhodný. Tzn. především jméno záznamu, název mapy, počet hráčů v zápase, výsledek, délku hry a pro oba hráče jejich APM a MMR statistiky. Na základě těchto informací pak program vyhodnotí, zdali je soubor použitelný či nikoliv. Podrobné podmínky určení vhodného záznamu jsou uvedené v kapitole 6.1.1. Rovněž jsou také ignorovány záznamy, které jsou poškozené. Extrahování proběhne po zavolání funkce `start` v modulu `py_replay_info_parser.py`. Po dokončení běhu je na standardní výstup zobrazeno shrnutí, tedy statistiky o zpracovaných souborech, jejich počet a poškozené a vyfiltrované soubory. Dále jsou do příslušných výstupních souborů zapsány informace ohledně všech vybraných záznamech, které jsou vhodné pro další zpracování. Program předpokládá umístění záznamů v adresáři *all* uvnitř adresáře *Replays*.

5.3.2 Přesun záznamů

Pro lepší přehlednost je třeba přesunout záznamy z balíků souborů do nového adresáře, jelikož jich obvykle může být velké množství (statisíce). Přesun souborů je proveden funkcí `start` v `py_replay_mover.py` a typicky tak navazuje na předchozí krok. Program je nutné spustit s přepínačem `--mode move`. Funkce projde všechny výstupní soubory vygenerované v předchozím kroku a překopíruje soubory, jejichž jména jsou uvedeny v daných souborech, do nové složky *tvt*. Ta bude tedy obsahovat veškeré soubory, které budou v následujícím kroku zpracovány.

5.3.3 Zpracování záznamů

Poslední krok, který navazuje na oba předchozí, je konečné zpracování vhodných záznamů z her. Zpracování je možné spustit ve více vláknech a urychlit tak dobu běhu programu. K tomu slouží modul `py_replay_parser_runner.py`, jenž spouští jednotlivá vlákna, která se starají o zpracování záznamů. Obsluhu samotného zpracování řeší třída `PyReplayParser`, ta volá hlavní metodu `process_replay`, která zpracovává jednotlivě záznamy.

Podstata zpracování tkví v transformaci interní reprezentace záznamů z klienta hry na vrstvy rysů, které je poté možné dále využít v PySC2. Tato transformace probíhá pomocí kontroléru hry, který je dostupný skrze PySC2. Kontrolér interaguje s prostředím a pomocí metody `transform_obs` konvertuje snímky hry na vrstvy rysů. Dále jsou zpracovávány provedené akce a jejich argumenty. Tyto údaje jsou pak spojeny do slovníku, který obsahuje záznam o jednom stavu hry. Tedy reprezentaci prostředí pomocí vrstev rysů a odpovídající akci, která byla provedena hráčem na základě tohoto stavu. Tyto záznamy jsou pak zapisovány do *pickle* souborů. Při transformaci se vyskytuje velké množství duplikátních akcí, jelikož je zpracování příkazů od hráče odlišné. Typicky se může stát, že hráč některé příkazy zadá velmi rychle za sebou (např. několik rychlých kliků myší na totéž místo), tyto příkazy se pak objeví v jednom časovém snímku hry (přibližně 60 ms) vícekrát. Takovéto duplicitní záznamy jsou ignorovány a je ponechán pouze jeden z nich. Při konverzi snímků se také mohou vyskytnout chyby v záznamech, které blokují jejich zpracování. Chybné záznamy jsou pak přeskočeny a jejich název je uložen do souboru *invalid_replays.pickle*. Pokud by se

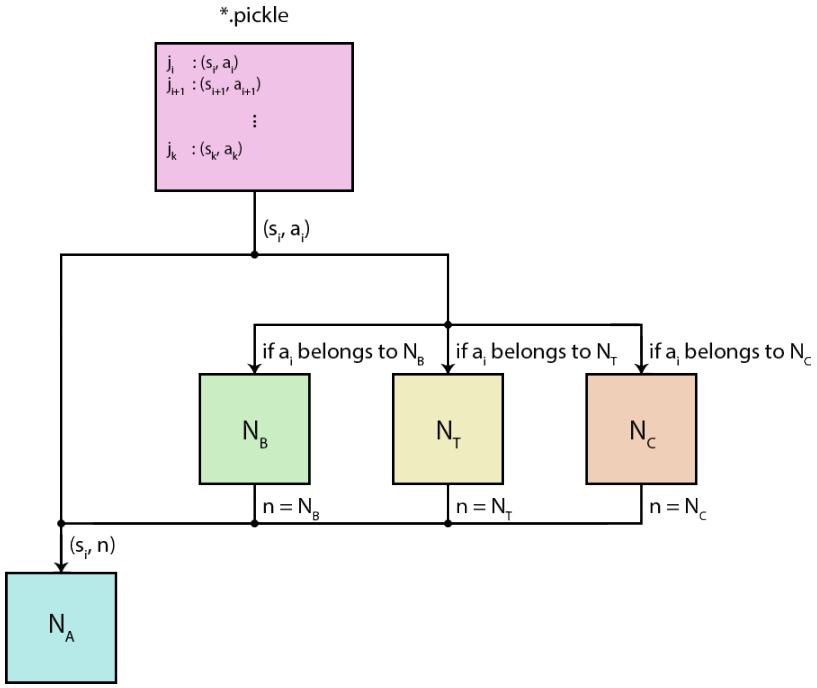
pak znovu narazilo na některý z těchto záznamů, je ignorován. Takto poškozené záznamy byly pozorovány především u verze z klienta 4.0.2.

5.3.4 Technika učení

Po konverzi záznamů na vrstvy rysů, která byla popsána v předchozí podkapitole, je možné začít s procesem učení. To probíhá formou metody učení s učitelem, kdy síť odhaduje třídy, do kterých mohou akce patřit, a následně jsou tyto odhady srovnávány se skutečnými hodnotami.

Pro tento účel byl vytvořen program *PyBot*, ten umožňuje, jak učení modelu ze záznamů, tak pomocí posilovaného učení. Pro spuštění učení ze záznamů je nutné při spuštění specifikovat přepínač `--supervised`, který spustí učení z vyfiltrovaných záznamů. Program je možné spustit také ve více vláknech při zvolení přepínače `--threads`. Zrychlí se tak učení neuronových sítí. Kontrolu nad zpracováním obstarává `py_supervised_runner.py`, který volá funkci `run_master_thread` pro spuštění zapisovacího vlákna a funkci `run_thread`, která spouští jednotlivé vlákna s třídou `PySupervisedLearner`. Zapisovací vlákno se stará především o shromažďování a ukládání celkových statistik průběhu učení mezi ostatními vlákny.

Při spuštění programu dojde k sestavení modelů sítí metodami `assemble_network` a `assemble_arbiter_network` třídy `PyNetwork` pro herní a řídící síť. Poté jsou vytvořeny výpočetní grafy, definované operacemi skrze Tensorflow, pomocí metod `assemble_graph` a `assemble_arbiter_graph` třídy `PySupervisedLearner`.



Obrázek 5.5: Znázornění učení ze zpracovaných záznamů. Model čte dvojice stavu hry a odpovídající akce (s_i, a_i) z *pickle* souborů. Stav a akci přivede na vstup hráčské sítě N_B, N_T, N_C dle toho, jaké síti akce náleží. Na vstup řídící sítě N_A je pak přivedena dvojice stavu hry a hráčské sítě, která danou akci zpracovala (s_i, n).

Samotná třída `PySupervisedLearner` se stará o učení modelu, které probíhá po zavolání metody `start_learning_from_replay`. Tato metoda načte dostupné zpracované záznamy a jednotlivě načítá stavy hry z vygenerovaných *pickle* souborů, čímž vlastně zpětně přehrává daný zápas tak, jak se odehrál. Systém učení je inspirovaný trénováním agentů pomocí algoritmu A3C. Učení probíhá v mini-dávkách (angl. mini-batch) a asynchronní aktualizaci parametrů globální (master) sítě. Každé vlákno zpracovává samostatně svoji sadu záznamů, přičemž při každém naplnění mini-dávky jednotlivými vzorky, resp. stavy hry a odpovídajícími akcemi, dojde k aktualizaci vah příslušné sítě. Model obsahuje mini-dávky pro každou neuronovou síť zvlášť. Celkem tedy používá čtyři odlišné, tři pro podsítě a jednu pro řídící síť.

Při načtení dvojice (s_i, a_i) ze souboru, kde s_i je stav prostředí a a_i je příslušná zvolená akce, přičemž $i < t$, kde t je finální stav, dojde k vyhodnocení, zdali akce a_i patří do akcí B příslušící síti N_B, nebo do akcí T příslušící síti N_T, nebo do akcí C příslušící sítí N_C. Poté se příslušné síti přiřadí daný stav a akce, a řídící sítí se přiřadí dvojice stav a odpovídající značení sítě, do které daná akce patří, např. (s_i, N_T). Základní funkčnost tohoto procesu je vyobrazena diagramem 5.5.

Některé akce se však mohou mezi sítěmi překrývat, tedy mohou patřit do více sítí, v tom případě je daná akce přirazena naposledy aktivní síti, dokud nedojde k přepnutí na síť jinou. V tom případě se k dané akci, která patří do jiné sítě, přidruží i akce jí předcházející. Jinými slovy se toto rozřazení snaží napodobit chování hráče. Uvažujme například, že hráč bojuje s jednotkami na jiném konci mapy, než má základnu. V okamžiku, kdy má trochu volna, se přesune zpět na svoji základnu a zadá příkazy pro vytrénování dalších jednotek a poté se vrátí zpět k bojujícím jednotkám. Pokud tedy použijeme výše uvedené rozřazení akcí, dojde k tomu, že dokud hráč bojuje se svými jednotkami, počítají se společné akce (typicky posouvání kamery) do sítě N_C, v okamžik, kdy hráč chce postavit nové jednotky, se k současné akci připočítá i předchozí akce tak, že oboje patří do sítě N_T. Před zadáním příkazu k tréninku došlo typicky k posunu kamery zpět na základní bázi hráče.

Řídící síť N_A se tedy snaží odhadnout záměr hráče, co bude na základě stavu (fázi) hry dělat. Zdali chce trénovat jednotky, stavět budovy, či útočit s jednotkami. Jednotlivé podsítě N_B, N_T, N_C se pak snaží odhadovat akce, které hráč konkrétně vykoná, tzn. jakou budovu postaví, kterou jednotku vytrénuje či kde bude útočit.

5.4 Trénování agenta

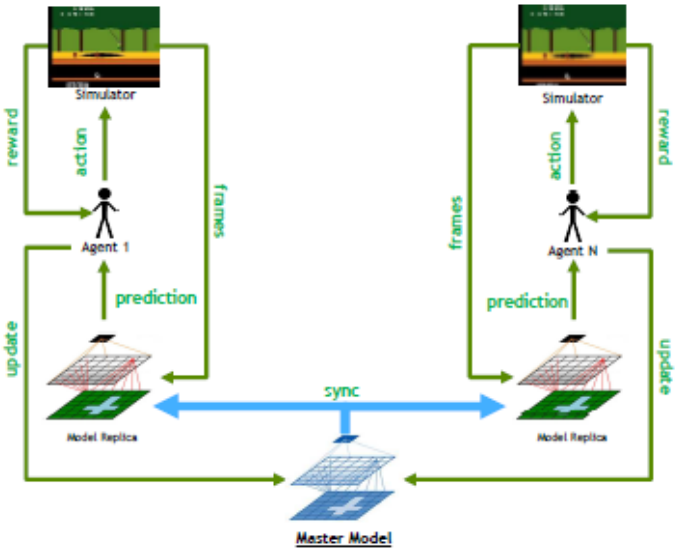
Následující podkapitola popisuje algoritmus A3C a jakým způsobem byl využit pro trénování agenta. Dále je podrobně popsána implementace a princip trénování.

5.4.1 Aktér-kritik

Patrně nejpopulárnější varianta z rodiny aktér-kritik algoritmů je A3C (angl. asynchronous advantage actor-critic) [20]. Za jeho vytvořením stojí opět tým DeepMind, který pomocí něj překonal ve výkonnosti hluboké Q-sítě (angl. deep Q-Networks), které předtím excelovaly v hraní Atari her [21].

Základem A3C je jedna globální síť a několik agentů, kteří s ní komunikují. Globální síť slouží jako centrální úložiště vah, zatímco každý agent interaguje se svojí kopií prostředí, viz obr. 5.6. Poté co agenti nasbírají dostatek vzorků, nebo při dosažení koncového stavu, provedou výpočet gradientů dle loss funkce s ohledem na parametry své lokální sítě a poté svým výpočtem aktualizují váhy globální sítě. Své váhy synchronizují s váhami globální sítě

a opět pokračují ve sbírání vzorků. Nasbírané sadě vzorků se říká tzv. mini-dávka. Velikost mini-dávky byla nastavena na 40 vzorků.



Obrázek 5.6: Grafické znázornění algoritmu A3C. Základ tvoří globální síť, která slouží pro synchronizaci vah napříč agenty. Jednotliví agenti pak interagují s prostředím a vzájemně aktualizují váhy globální sítě, se kterou se pak opětovně synchronizují [2].

Paralelizace agentů přináší několik výhod. Dvě největší jsou snížení korelace vstupů a zvýšení množství odvedené práce. Každé prostředí, se kterým agent právě komunikuje, se typicky nachází v jiném stavu a agent tak má k dispozici jiné akce a získává jiné odměny. Globální síť tak disponuje poznatky od všech agentů a zvyšuje rychlost učení a robustnost sítě a snižuje varianci. Druhá výhoda je snížení korelace vstupů, a tím zvýšení odolnosti proti přeučení.

Pro aktér-kritika byl použit následující gradient:

$$\nabla_{\theta'} \log \pi(a_t \mid s_t; \theta') A(s_t, a_t; \theta, \theta_v) + \alpha \cdot (R - V(s_t; \theta'_v))^2 + \beta(\nabla_{\theta'} H(\pi(s_t; \theta'))) \tag{5.5}$$

Kde π je strategie, α je koeficient funkce ohodnocení, β je koeficient entropie, θ a θ_v jsou váhy (parametry) globální sítě, θ' a θ'_v jsou váhy lokální sítě, s_t je stav, a_t je akce, R je očekávaný návrat, V je funkce ohodnocení, $H(\pi)$ je entropie strategie, která podporuje prohledávání a brání konvergenci k suboptimální strategii, přičemž $H(\pi(s))$ je definována následovně:

$$H(\pi(s)) = -\sum_{i=1}^{n} \pi(s_i) \log(\pi(s_i)) \tag{5.6}$$

Dále, kde $A(s_t, a_t; \theta, \theta_v)$ je funkce prospěchu definována jako:

$$A(s_t, a_t) = Q(a_t, s_t) - V(s_t) = R_t - V(s_t) = \sum_{t=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k}) - V(s_t) \tag{5.7}$$

Kde s_t je stav, a_t je akce, V je funkce ohodnocení, R je očekávaný návrat, γ je redukční faktor a k definuje počet kroků, přičemž bylo zvoleno $\gamma = 0.99$ a $k = 8$.

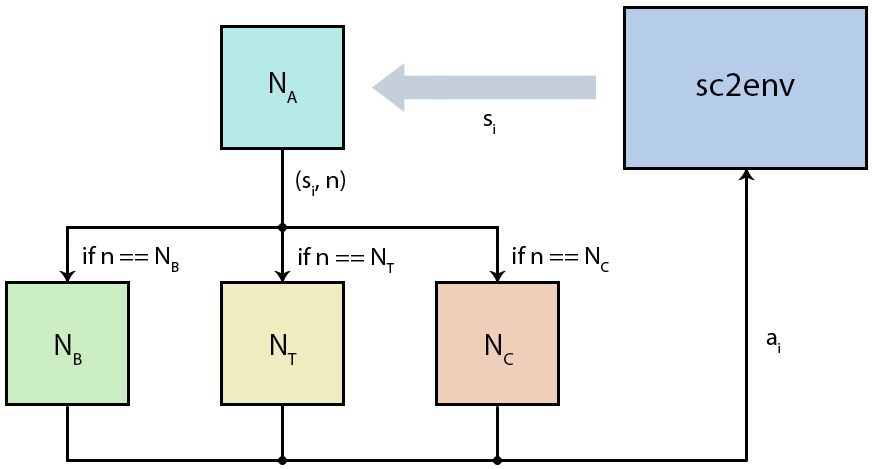
Při inicializaci programu dochází k sestavení modelů globálních sítí pomocí metod třídy `PyNetwork` `assemble_network` pro herní sítě a `assemble_arbiter_network` pro řídící síť. Poté následuje inicializace jednotlivých agentů pomocí metod třídy `PyAgent` `assemble_graph` pro herní sítě a `assemble_arbiter_graph` pro řídící síť. Tyto metody sestaví strom operací definovaných skrze Tensorflow, které slouží jako hlavní tělo pro výpočet gradientů a aktualizací vah modelu.

5.4.2 Trénink

Druhou část *PyBota* tvoří implementace algoritmu A3C, který pomocí posilovaného učení trénuje dále síť. Koncept je podobný jako u učení ze záznamů. Trénování je možné spustit ve více vláknech pomocí funkce `run_thread` při specifikování přepínače `--threads`, která spravuje jednu instanci třídy `PyAgent`. Zapisovací vlákno je voláno funkcí `run_master_thread` a opět slouží především jako společný shromažďovač statistik a k ukládání vah master sítě.

Při spuštění nového vlákna se inicializuje prostředí Starcraftu s požadovaným nastavením (např. jméno herní mapy, zdali se má prostředí vykreslovat skrze PySC2 apod.). Spuštění hry nastává při zavolání metody `reset`, která inicializuje hru. Třída `PyAgent` taktéž zavolá metodu `reset`, která synchronizuje váhy globálních sítí s těmi lokálními.

V dalších krocích je opakovaně volána metoda prostředí `step`, která vrací nový snímek (stav) hry. Třída `PyAgent` pak nejprve zavolá metodu `decide`, která skrze řídící síť rozhodne na základě stavu hry, jaká podsíť bude aktivní. Následně se volá metoda `step`, která zpracuje snímek hry a vygeneruje pro příslušnou síť odpovídající akci a případně parametry, které agent vykoná, viz diagram 5.7 znázorňující tento proces.



Obrázek 5.7: Model řízení při komunikaci agenta s prostředím. Řídící síť N_A obdrží stav hry s_i v kroku i od prostředí *sc2env*, na základě kterého se poté rozhodne, jaká z herních sítí N_B, N_T, N_C bude aktivní. Daná síť poté provede akci a_i a tento proces se opakuje, dokud se prostředí nenachází v koncovém stavu.

Zpracování stavu hry probíhá pomocí třídy `PyPreprocess`, která odděleně zpracovává vrstvy rysů obrazovky skrze metodu `preprocess_screen`, vrstvy rysů minimapy skrze metodu `preprocess_minimap` a dále statistiky o hře (např. počet natěžených surovin, aktuálně vybraná jednotka, odehraný čas apod.) pomocí metody `preprocess_stats`. Mezi hodnotami jednotlivých atributů mohou být velké rozdíly v řádech, což by mohlo negativně ovlivnit váhy sítí, např. atribut *player_relative* nabývá pouze hodnot $\langle 0; 4 \rangle$, zatímco atri-

but *unit_hit_points* může nabývat hodnot $\langle 0; 1600 \rangle$. Z tohoto důvodu jsou všechny atributy normalizovány předpokládanou nejvyšší hodnotou, jaké mohou nabývat.

Agent provede zvolenou akci a obdrží příslušnou odměnu. Stav hry (vrstvy rysů), provedená akce, parametry a odměna jsou společně uloženy do patřičné mini-dávky příslušné sítě skrze metodu `store`. Pokud je mini-dávka pro podsíť, resp. řídící síť, naplněná, zavolá se z ní privátní metoda `_update`, resp. `_update_arbiter`, která na základě loss funkce vypočítá gradienty a aktualizuje váhy sítě.

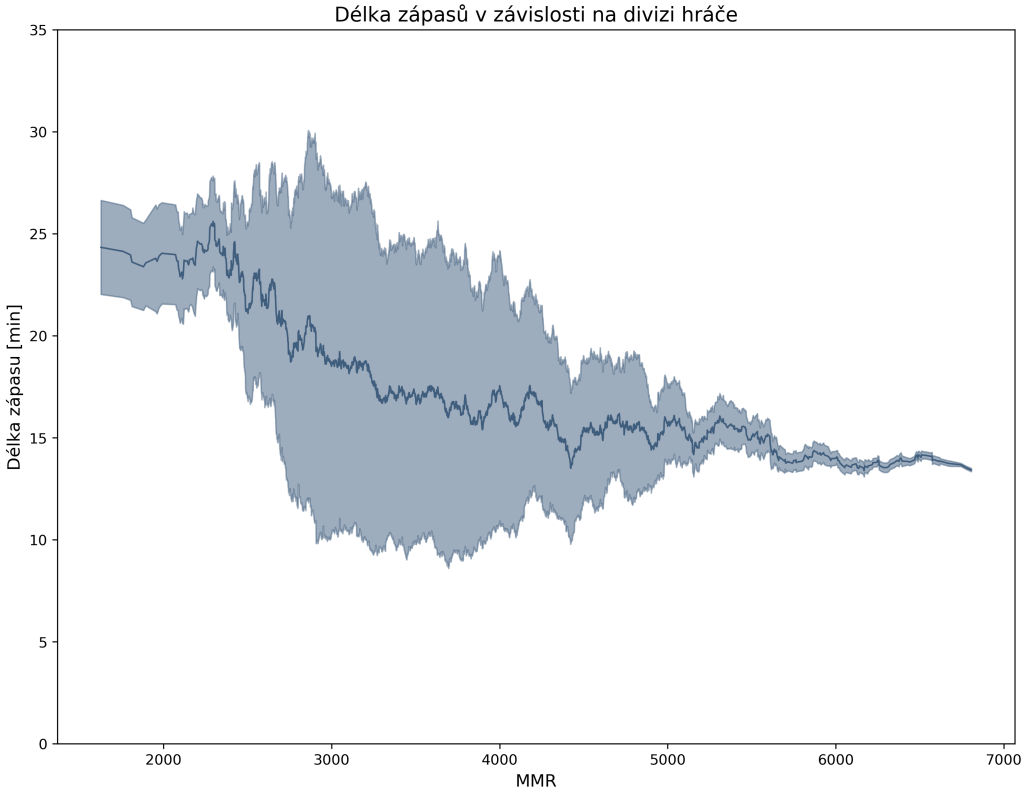
Váhy globální sítě jsou periodicky sbírány (přibližně každou minutu) pomocí zapisovacích vláken, jak v případě tříd `PySupervisedLearner`, tak i `PyAgent`, a ukládány. V případě učení ze záznamů se dále počítá přesnost sítí a v případě trénování agenta skóre a výsledky z odehraných her. Dále jsou také periodicky ukládány aktualizované váhy sítě a zapisovány do tzv. *checkpoint* souborů pomocí Tensorflow, a to zvlášť pro každou z podsítí a pro řídící síť. Pro každou ze sítí se individuálně počítá tzv. globální krok, který slouží pro správné zapisování statistik, lze tak jednoduše každou ze sítí trénovat zvlášť. Proces trénování nebo učení je možné přerušit a dosavadní pokrok bude zapsán do zmiňovaného *checkpoint* souboru včetně hodnoty globálního kroku. Při novém spuštění procesu je tak možné uložený soubor opětovně nahrát a pokračovat ve stejném místě.

Kapitola 6

Testování

Tato kapitola se v první části zabývá vyhodnocením učení ze záznamů. V části druhé pak bude rozebráno trénování pomocí algoritmu aktér-kritků a vyhodnocení úspěšnosti modelu.

Trénování probíhalo pomocí GCE (Google Compute Engine) na několika linuxových strojích běžících na Ubuntu ve verzi 16.04. Byly odzkoušeny různé kombinace počtu procesorů od 4 až 32, avšak vzhledem k nákladům na provoz daných stanic a délce trénování byla většina modelů trénována na strojích 8 – 16 procesory a typicky alespoň 16 – 32 GB RAM. Od určitého počtu vláken byl již přísun vetší paralelizace sporný vzhledem k implementaci v Pythonu s využitím pouze procesorů bez grafických karet a nedistribuované verze Tensorflow.



Obrázek 6.1: Délka zápasů vzhledem k MMR hráče. Je patrné, že s rostoucím MMR dochází se snižování délky her. Hráči se tak snaží spíše o agresivnější taktiku než defenzivní.

6.1 Učení ze záznamů

V první části této kapitoly bude rozebráno učení modelu ze záznamů. Nejprve bude popsán dataset, na kterém bylo učení prováděno a následovat bude zhodnocení modelu.

6.1.1 Dataset

K natrénování modelu bylo použito celkem 11032 záznamů ze dvou verzí klienta, konkrétně 4083 z verze 3.16.1 a 6949 z verze 4.0.2. Záznamy byly vyfiltrované z balíků, které obsahovaly dohromady přes 200 tisíc souborů, z nichž bylo necelých 30 tisíc TvT záznamů (rasa Terran vs. Terran), tedy přibližně 14,6 % z celkového počtu. Dle níže uvedených kritérií pak bylo vybráno necelých 12 tisíc záznamů, které byly použitelné. Dalších zhruba 800 záznamů z nich bylo poškozených a nebylo je tak možné zpracovat. Z celkového počtu dostupných záznamů tedy bylo použito přibližně 5,4 %. Skutečná délka všech zpracovaných zápasů je přes 3200 hodin reálného času. Podrobnější rozpis je k nahlédnutí v tabulce 6.1.

Tabulka 6.1: Tabulka počtu použitých a zamítnutých záznamů z her

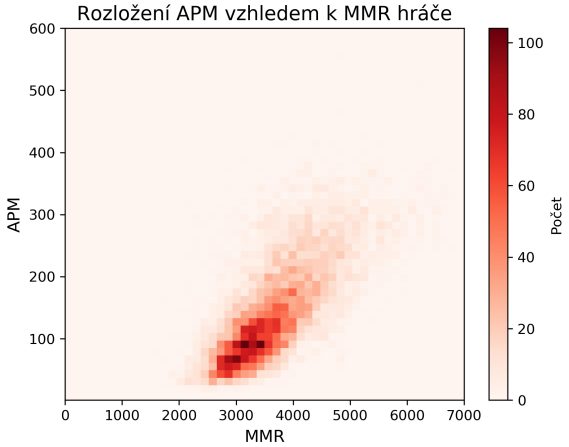
Verze hry	Celkem	Počet TvT	Vyfiltrované	Použité
3.16.1	64396	10353	4084	4083
4.0.2	139129	19339	7742	6949
Celkem	203525	29692	11826	11032

Kritéria pro výběr vhodných záznamů byla následující:

- Zápas se musel hrát ve formátu jeden na jednoho
- Oba hráči museli hrát za rasu Terran
- Záznam se odehrával z pohledu hráče č. 1
- Hráč č. 1 vyhrál zápas
- Zápas trval alespoň dvě minuty
- APM (počet akcí za minutu) bylo alespoň 10 pro oba hráče

Díky těmto podmínkám byly odfiltrovány nevhodné nebo poškozené záznamy. Balíky souborů obsahují každý zápas dvakrát, z pohledu hráče 1 a z pohledu hráče 2. Z celkového počtu TvT záznamů je tedy použita pouze polovina, a to ta, ve které zvítězí hráč č. 1. Dále jsou odstraněny zápasy, které netrvají alespoň dvě minuty, jelikož se jedná pravděpodobně o hry, ve kterých jeden z hráčů opustí předčasně hru, resp. téměř ihned po jejím zahájení se vzdá. Reálně není možné korektně odehrát zápas pod tento čas. S tímto souvisí i podmínka vyžadující, aby počet APM bylo alespoň 10 pro oba hráče. Při nižších hodnotách se jedná pravděpodobně o hráče, který téměř nebo vůbec nehraje, ale zároveň neopustil hru. Pro zajímavost je uvedena statistika délek jednotlivých her v datasetu vzhledem k MMR, viz graf 6.1. Přehled APM je na histogramu 6.2.

V úvaze bylo i využití hodnoty MMR (angl. matchmaking rating), což je hodnota, která udává dovednost hráče a úroveň jeho hraní. Čím vyšší je MMR, tím vyšší je umístění hráče v žebříčku všech hráčů a vypovídá více o tom, že je daný hráč ve hře dobrý, tzn. vyhrává zápasy a hraje racionálně. Naopak, čím nižší je MMR, tím je daný hráč ve hře horší. Podle



Obrázek 6.2: Rozložení APM vzhledem k MMR v datasetu. 2D histogram počtu akcí za minutu vzhledem k dovednostní úrovni hráče.

Tabulka 6.2: Přehled rozmezí hodnot MMR v jednotlivých divizích. Divize jsou seřazeny sestupně od nejvyšší po nejnižší. Tabulka je platná k sezóně 35 od ledna do května roku 2018 [4].

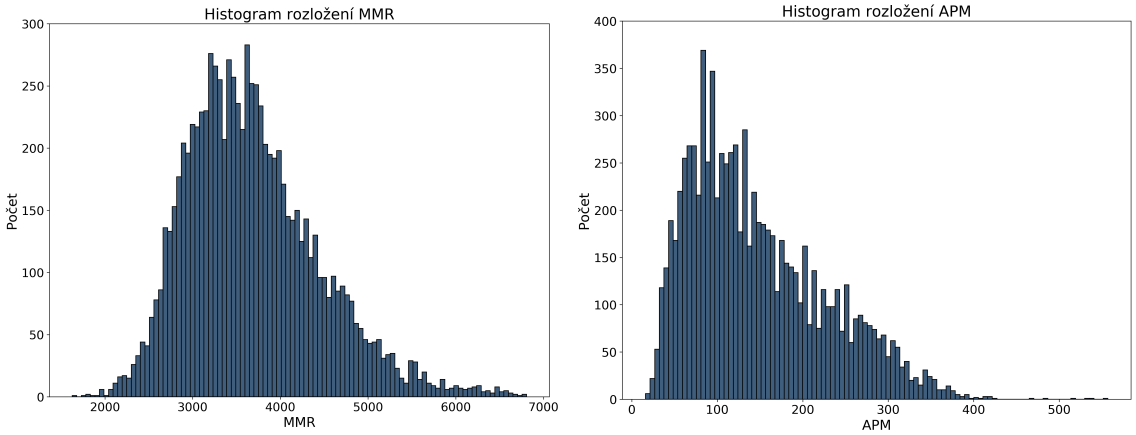
Divize	**Evropa**	**Severní Amerika**	**Korea**
Mistrovská	4520 – 5118	4600 – 5211	4800 – 5437
Diamantová	3640 – 4520	3680 – 4600	3800 – 4800
Platinová	3280 – 3640	3320 – 3680	3440 – 3800
Zlatá	3000 – 3280	3000 – 3320	3120 – 3440
Stříbrná	2440 – 3000	2440 – 3000	2600 – 3120
Bronzová	1822 – 2440	1829 – 2440	1963 – 2600

úrovně MMR lze pak rozdělit hráče do různých divizí, dle jejich dovednosti, např. bronzová, stříbrná či zlatá divize aj. Nicméně od tohoto návrhu muselo být upuštěno, jelikož velké množství záznamů, především z balíku verze 4.0.2, obsahuje nekorektně zaznamenané hodnoty MMR, a tudíž je není možné správně rozlišit. Průměrná hodnota MMR v datasetu, počítaná ze záznamů, ze kterých to bylo možné vyhodnotit, byla 3684, což zhruba odpovídá umístění na přelomu platinové a diamantové divize, tzn. dá se předpokládat, že hráči dosahující tohoto umístění hrají nadprůměrně a racionálně, viz přibližná tabulka rozdělení divizí 6.2. Tabulka je spíše orientační, jelikož se rozmezí pro každou herní sezónu přepočítávají na základě hodnot té předchozí. Navíc se jednotlivé meze liší pro různé světové regiony a u záznamů nelze přesně zjistit, kdy byly pořízeny či do jakého regionu daní hráči patřili. U hráčů z Evropy a Severní Ameriky je možné vidět téměř shodnou dovednostní úroveň, zatímco korejskou ligu hrají obvykle ještě mnohem zkušenější hráči. Průměrná hodnota počtu akcí za minutu v datasetu pak byla 146, viz histogramy 6.3.

Jako loss funkce byla pro jednotlivé sítě zvolena křížová entropie H (angl. cross-entropy) [14], která je definována jako:

$$H(P, Q) = -\sum_{x} P(x) \cdot \log Q(x) \tag{6.1}$$

Kde x je proměnná a P a Q jsou dvě různá rozdělení pravděpodobnosti. Křížová entropie měří vzdálenost (rozdíl) právě mezi těmito pravděpodobnostmi, přičemž P je skutečná pravděpodobnost a Q je současná.



(a) (b)

Obrázek 6.3: Histogram rozložení počtu APM a MMR hráčů v datasetu. Histogram a) zobrazuje rozložení MMR a histogram b) zobrazuje rozložení APM.

Po naplnění mini-dávek a vypočítání křížové entropie jsou vypočítány gradienty a váhy neuronové sítě aktualizovány. Každá instance třídy `PySupervisedLearner` uchovává svoji lokální kopii master sítě. Napříč jednotlivými instancemi pak dochází k asynchronním aktualizacím master sítě. Po aktualizaci jsou tyto váhy synchronizovány.

6.1.2 Průběh učení

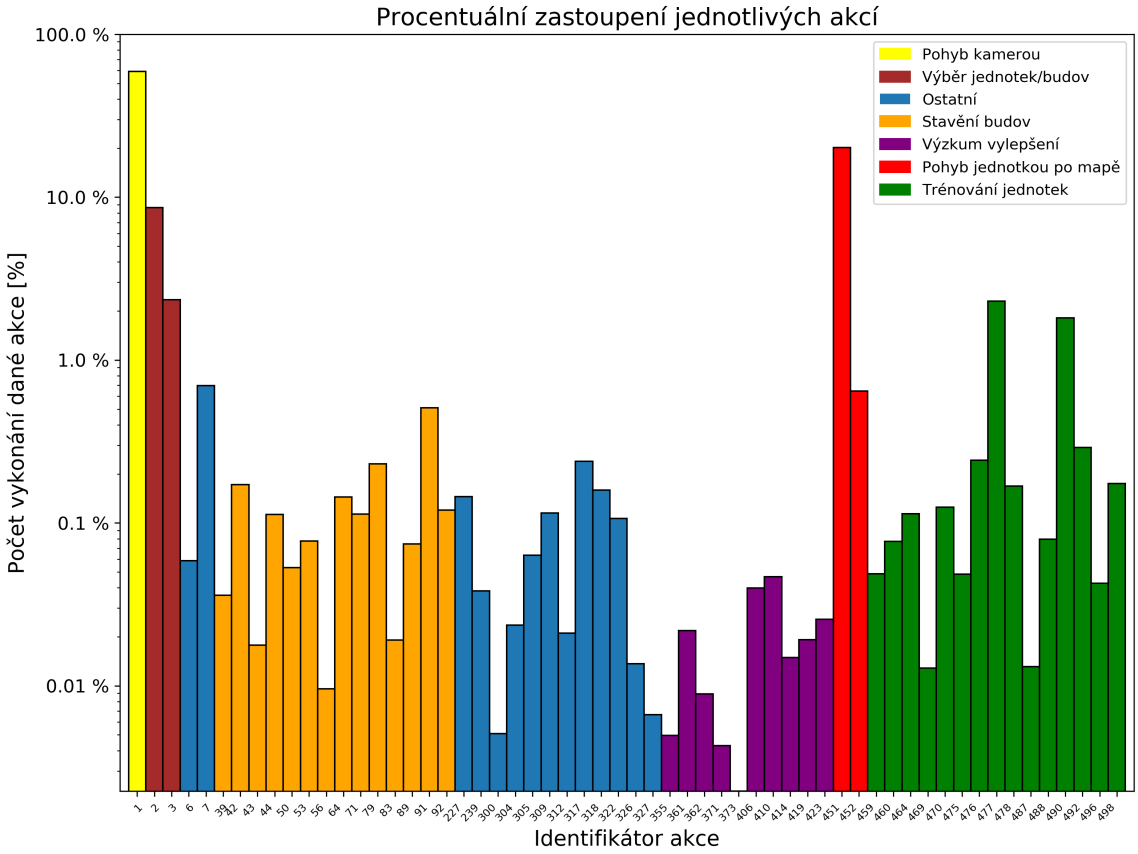
První fáze učení modelu probíhala pomocí učení ze záznamů her. Trénování ze zvolených záznamů trvalo zhruba 50 hodin čistého času. Velikost kroku při přehrávání záznamů byla zvolena na 8, což odpovídá zhruba dvěma možným akcím za sekundu. Při učení byla vyhodnocována přesnost sítě po celou dobu běhu.

Měření přesnosti probíhalo pomocí křížové entropie tak, že nejprve byla vygenerována akce danou neuronovou sítí a poté byla porovnána s akcí, která byla skutečně v záznamu provedena. Obdobně byla měřena přesnost odhadu argumentů akcí, resp. souřadnic na mapě a minimapě. Přesnost pro odhadované koordináty pro obrazovku a minimapu byla sloučena. Zapisovací vlákno sbíralo periodicky (přibližně každou minutu) statistiky o počtu správně a špatně uhodnutých hodnotách ze všech běžících vláken. Jejich průměrnou hodnotu pak zapisovalo do výstupního souboru. Naměřená přesnost tak odpovídá celkové průměrné přesnosti během času učení.

Měření se dělilo na dvě skupiny, a to přesnost pro řídící síť a přesnost pro akce a jejich argumenty pro trojici herních sítí dohromady.

Po dokončení učení byla řídící síť schopna s 77% úspěšností předpovědět, jaký typ akce se chystá hráč na základě stavu hry provést. Tedy, zdali chce stavět budovy, trénovat jednotky nebo útočit. Parametr učení byl zvolen $1 \cdot 10^{-5}$ a to na základě provedení několika pokusů s různými hodnotami. Průběh je vyobrazen na grafu 6.5.

Druhá skupina měřila přesnost pro odhadnutí přesné akce a argumentu a také přesnost pro prvních pět akcí a argumentů, tzn. zdali se skutečná akce či argument nacházely v pěti nejpravděpodobnějších akcích a jejich argumentech. Hodnota parametru učení byla pro



Obrázek 6.4: Procentuální zastoupení četnosti zvolených akcí v datasetu. Z grafu je patrné, že dominují akce typu pohyb kamery a přesun jednotek, což značně deformuje prostor akcí. Sety akcí (např. stavění budov nebo trénování jednotek) dosahují jen stěží 1 % – 2 %.

všechny tři sítě zvolena experimentálně $3 \cdot 10^{-4}$. Přesnou akci, kterou chce hráč provést, byla síť schopna odhadnout s 56% přesností po dokončení učení. To, že akce patří do pěti nejpravděpodobnějších, je pak síť schopna odhadnout s 93% přesností (viz grafy 6.6), avšak tato vysoká hodnota není tak překvapující, jelikož je v datasetu několik akcí, které v počtu provedení dominují nad všemi ostatními.

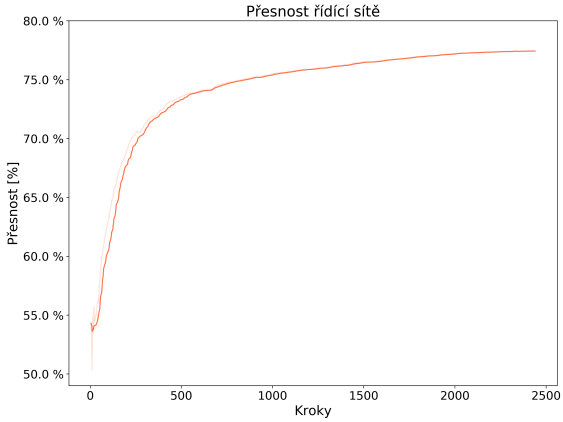
Přesnost odhadnutí správného prostorového argumentu k dané akci pak činí 28 %. Pro pět nejpravděpodobnějších argumentů je to pak 44 %, viz grafy 6.7.

Pro ilustraci se nabízí využít práci DeepMindu, kde přesnost pro odhadnutí akce činila necelých 38 % a pro pět nejpravděpodobnějších zhruba 88 %. Pro argumenty je to pak nejlépe necelých 11 %, resp. 26 %, pro argumenty vztahující se k prostorovým souřadnicím obrazovky, resp. minimapy. Pro pět nejpravděpodobnějších je to pak 22 % a 63 %.

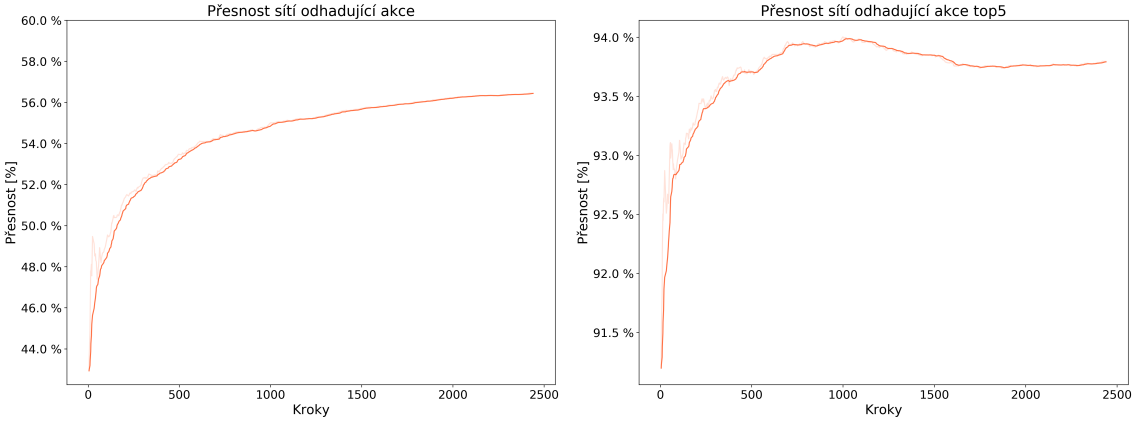
Vytvořený model tak dosahuje sice vyšší přesnost, ale nelze je efektivně porovnat, jelikož DeepMind ve své práci využívá širší spektrum akcí a větší prostorové rozlišení. Navíc rozděluje úspěšnost pro prostorové akce na obrazovku a minimapu.

6.1.3 Vyhodnocení naučeného modelu

Cílem práce bylo založit natrénování modelu na základě učení ze záznamů z her. Nicméně předpoklad, že se toto povede, byl spíše skeptický, jelikož v tomto neuspěl ani DeepMind se



Obrázek 6.5: Průběh učení řídící sítě.



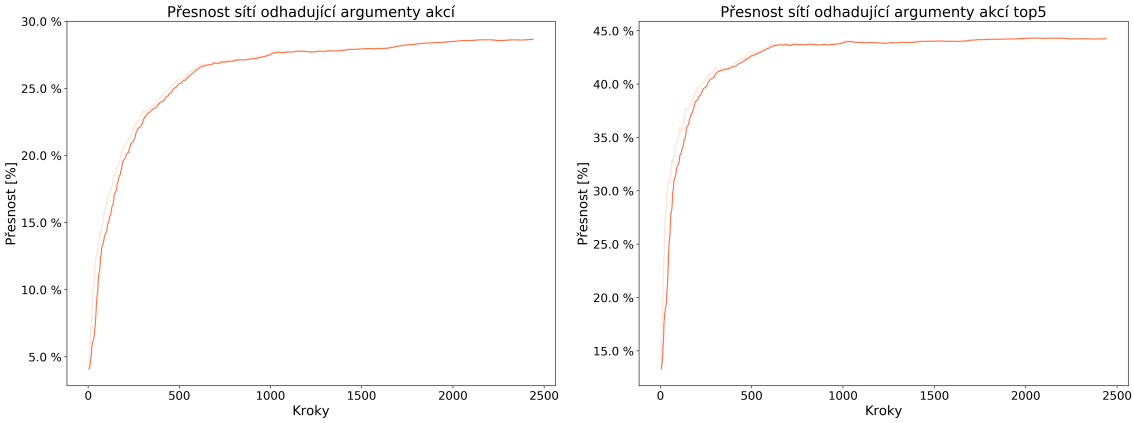
(a) (b)

Obrázek 6.6: Průběh učení sítě pro odhadování argumentů zvolené akce hráčem. Graf a) zobrazuje průběh přesnosti odhadnutí argumentů akce vykonané hráčem během učení. Graf b) pak zobrazuje průběh přesnosti odhanutí, že argumenty akce patří mezi pět nejpravděpodobnějších.

svým botem, kdy bot nebyl schopen vyhrát žádnou hru proti nepříteli ani po učení z 800 tisíc záznamů her.

Testování probíhalo na zjednodušené mapě *Flat64custom* o velikosti 64x64 s pěvně danou startovací pozicí. Bot i nepřítel měli k dispozici jednu oblast pro těžení surovin. Jednalo se o modifikaci mapy *Flat64*, kterou vytvořil DeepMind. Jako soupeř byl zvolen zabudovaný bot ve hře na nejlehčí úroveň bez dalších omezení. Časový limit mapy byl nastaven na 30 minut, po kterých byla stanovena remíza a FoW bylo zapnuté.

Podsíť N_B měla k dispozici 18 akcí, které mohla využívat. Patřilo mezi ně především stavění budov, označování jednotek a posun kamery. Podsíť N_T mohla vybírat z 34 akcí, mezi něž se řadilo trénování jednotek, výzkum vylepšení, označování jednotek a posun kamery. Poslední podsíť N_C vybírala z 11 akcí, ve kterých bylo především posouvání s jednotkami, posun kamery a používání speciálních akcí vojenských jednotek. Rozpis jednotlivých akcí v této fázi je k dispozici v příloze B.



(a) (b)

Obrázek 6.7: Průběh učení sítě pro odhadování zvolené akce hráčem. Graf a) zobrazuje průběh přesnosti odhadnutí akce vykonané hráčem během učení. Graf b) pak zobrazuje průběh přesnosti odhanutí, že akce patří mezi pět nejpravděpodobnějších akcí.

Bot odehrál více než 2500 her. Nicméně z toho pouze 4 hry vyhrál a ve 13 z nich remizoval, viz grafy 6.8. Průměrné skóre se pohybovalo okolo 6380 bodů během her. Tento výsledek se tak velmi blíží závěrům týmu DeepMind. Vytvořený bot se byl schopen naučit stavět základní budovy a vytrénovat několik vojenských jednotek během hry. Toto však bohužel nestačilo na osvojení si nějaké základní strategie, která by byla dlouhodobě úspěšná a které by se držel.

Pozitivní zprávou je fakt, že rozdělení problému hraní na podproblémy, umožnilo botovi hrát o něco lépe, několik her vyhrát a v dalších remizovat. A dokonce nahrát vyšší skóre během her, přestože bot DeepMindu měl větší předpoklady pro to nahrát vyšší skóre, jelikož se zápas odehrával na větší mapě. Remíza nastala tehdy, pokud se do daného časového limitu nestihl bot s nepřítelem navzájem zničit. Nemohlo však nastat to, že by bot odletěl s budovami mimo dosah nepřítele, jako se to dělo u bota DeepMindu, jelikož měl tuto akci zakázanou.

Nicméně se však potvrdily skeptické odhady, že učení ze záznamů nebude dostatečné na to, aby byl program schopný hrát celé hry. A to bohužel ani s tím, že bylo hraní rozděleno na méně komplexní části a za zjednodušených podmínek. Důvodů proč záznamy nejsou dostatečné je hned několik:

- Deformace posloupnosti akcí. Lidské hraní má jednu velkou nevýhodu a to, že je pomalé. Záznamům dominují dvě akce, posun kamery a žádná operace. Tento fakt velmi mate učení neuronových sítí. V návrhu byla operace pro nicnedělání zcela ignorována, jelikož byl model velmi zkreslený. Mezi podstatnými akcemi hráče tak může být až několik set „zbytečných“ akcí. S tímto souvisí i robustnost akcí. Výše zmíněné úkony se používají prakticky ve všech situacích od stavění a trénování až po útočení a tvoří tak „výplň“ mezi důležitými akcemi, které v počtu provedení dosahují dohromady jen několika málo procent.

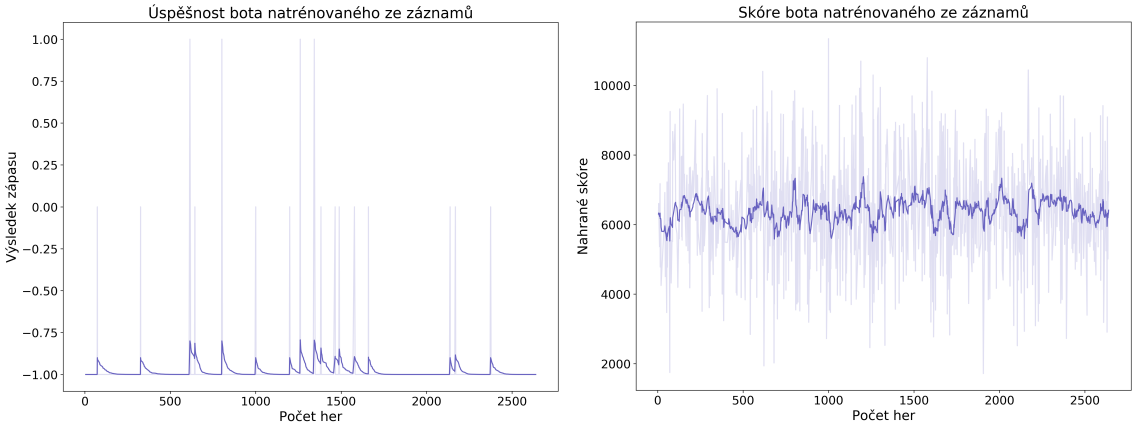
- Časová prodleva. Prakticky všechny akce ve hře mají velkou časovou prodlevu, a to i když pomineme fakt, že člověk hraje pomalu. Od pohybu po mapě nebo trénování až po stavění. Trénování jednotek trvá od 17 sekund a 25 sekund (pro nejzákladnější jednotky) až po 90 sekund. Totéž platí pro budovy, kde se toto rozmezí pohybuje od

25 sekund až po 100 sekund. Mezi zadáním příkazu a vytvořením daného objektu ve hře tak v případě bota uběhne až několik set snímků, resp. stavů hry. Hledat spojitosti mezi těmito věcmi je pak pro sítě velice složité a je patrné, že i v případě kdy obsahují rekurentní buňky.

- Záměr. Lidští hráči obvykle hrají s nějakou rozmyšlenou strategií a podle toho vykonávají určité akce, které jsou velmi důležité a na ostatní akce neberou tolik zřetel. Jinými slovy, např. čekají, než se dostaví nějaká jednotka nebo budova a mezitím mohou jen náhodně klikat po mapě nebo na budovy anebo přesouvat jednotky, i když tyto akce nemají praktický význam. Jelikož síť nezná hráčův záměr, tak je tímto faktem opět ovlivněná.

Dalším důvodem, proč systém neuspěl, je, že nebyl schopen hrát rychle. Starcraft je hra založená na rychlosti, tzn. cílem je co nejrychleji postavit základní vojenské budovy a začít trénovat jednotky, což znamená i rychle těžit suroviny. Síť nebyla schopná se naučit provádět tyto akce dostatečně rychle, což znamená, že ve chvíli, kdy bot stavěl prvních několik budov či jednotek, podnikal nepřátelský bot již útok na základnu.

Obzvláště problematické bylo sbírání suroviny *vespene*. Jedná se o speciální ložiska plynu, která hráč musí vytěžit. U každé základny se nacházejí tyto ložiska dvě a k těžení je třeba na každém postavit speciální budovu. Navíc je třeba k tomuto úkonu přiřadit několik těžebních jednotek, avšak ne příliš mnoho, jelikož jednak se těžba již poté nezvyšuje, a naopak upadá těžba minerálů. *Vespene* je potřebný prakticky ke všem pokročilejším úkonům ve hře, tzn. stavba lepších budov, silnějších jednotek, provádění výzkumu apod. Bez ní je bot odkázaný na stavbu pouze základní vojenské budovy a trénink základní vojenské jednotky.



(a) (b)

Obrázek 6.8: Výsledky bota natrénovaného ze záznamů. a) Výsledek zápasů, kde −1 značí prohru, 0 remízu a +1 výhru. b) Dosažené skóre během her.

Právě v neschopnosti dobře si osvojit obzvláště tyto dva zmíněné herní vzorce vedou k porážkám bota v drtivé většině případů.

Ukázalo se tedy, že (nejen) strategické hry, obzvláště odehrávající se v reálném čase, jsou pro neuronové sítě velmi komplikovaná záležitost pro naučení, což platí i tehdy, když mají k dispozici tisíce hodin záznamu, ze kterého se mohou učit.

Nicméně se projevilo, že díky učení ze záznamů jsou sítě schopny utvořit si snáze „povědomí“ o prostorovém uspořádání mapy, než při „čisté“ síti trénované na minihrách. Síť tedy

chápe systém souřadnic obrazovky a minimapy a pozice, kde se jaké jednotka či budova nachází, což jí umožňuje lépe a rychleji označovat jednotky a budovy kliknutím na obrazovku. K tomuto závěru bylo možné dojít při porovnání doby učení modelu ze záznamů a doby trénování modelu pomocí posilovaného učení pouze na minihrách, které dal k dispozici DeepMind. V druhém případě totiž trvalo několik desítek hodin, aby se byl agent schopen naučit hrát danou minihru ale přitom neporozuměl obecnějšímu chování hry. Minihry totiž disponují velmi omezeným herním prostorem a jen několika různými jednotkami, zatímco na záznamech má model možnost opakovaně vidět nejrůznější jednotky a budovy od hráčů na odlišných částech mapy a v rozličných situacích.

Vzhledem k tomuto faktu bylo vyzkoušeno další dotrénovaní sítě pomocí posilovaného učení, které by mohlo přispět k lepším výsledkům. Tento postup je popsán v následující podkapitole.

6.2 Trénování pomocí posilovaného učení

Další trénování navazuje na předchozí učení ze záznamů, tzn. využívá již částečně naučené sítě a není tak nutné je trénovat znovu. Záznamy tak posloužily jako jakési „zahřátí“ daných sítí. Trvalo by velmi dlouho, pravděpodobně nejméně stovky hodin, navíc s nejistým výsledkem, pokud by byly sítě trénovány znova. Takto bylo možné využít již některé naučené znalosti. Nicméně zároveň se sítě musely vypořádat se špatně naučenými akcemi, tedy především s pohybem kamery, těžením a trénováním jednotek, které velmi ovlivnily jejich chování a bylo nutné tyto akce „přeučit“.

Změnou je přidání čtvrté herní sítě N_G označené jako *GatherNetwork*, která se zabývá pouze těžením surovin s cílem natěžit co nejvíce vzácného materiálu *vespene*, který činil botovi velké problémy. Tato síť je aktivní pouze první dvě minuty na začátku hry, ve kterých musí připravit vše potřebné, tj. postavit budovy na těžení, vycvičit těžební jednotky a přiřadit jim práci. Během průběhu hry ji není možné znovu aktivovat. Využívá kopii vah sítě N_B a nebylo tak nutné ji zvlášť učit.

Dále bylo sníženo množství akcí, se kterými mohou jednotlivé sítě hrát. Ukázalo se, že natrénovat síť k tomu, aby byla schopna používat efektivně pohyb kamery je velmi problémové. Proto bylo rozhodnuto, že ovládat kameru bude pouze síť N_C, která útočí na nepřítele a je to pro ni nezbytné. Ostatní sítě pak hrají s kamerou fixní, tzn. disponují pohledem pouze na základnu a přilehlé okolí, na kterém mohou stavět anebo trénovat jednotky.

Nově bylo tak rozdělení akcí následovné, síti N_T byly odebrány akce, které umožňovaly zkoumat vylepšení, k tomuto kroku bylo přistoupenu vzhledem k faktu, že všechna vylepšení stojí spoustu materiálu *vespene*, který je velmi problematické pro agenta natěžit a může ho raději využít pro stavbu jednotek. Navíc je jejich efekt marginální. N_T tedy dohromady využívá 17 akcí. Síti N_B byly ponechány budovy, které produkují jednotky, nebo jsou nezbytné pro odemknutí dalších budov, nemůže tak stavět například další základny, jelikož je mapa situována tak, že to není třeba. K dispozici má tak 9 akcí. Navíc je omezena určitým limitem pro budovy. Nemůže tedy stavět více budov, než kolik reálně využije, např. budovy *SupplyDepot* zvyšují hranici, která udává, kolik je možné cvičit jednotek. Tuto hranici je však možné zvýšit jen do určité míry a pak již další budovy nemají žádný význam. Síť má proto nastaveno, kolik takových budov může postavit. Síť N_G nyní obstarává těžení surovin na začátku hry a může trénovat těžební jednotky. Celkem má k dispozici 6 akcí. Síti N_C zůstávají akce prakticky neměnné. Nové rozložení akcí v této fázi je k náhledu v příloze B.

Osvědčilo se také přidat parametr ϵ, kde $\epsilon = 0.1$, se kterým agent vykoná s určitou pravděpodobností náhodnou akci, což je typické pro tzv. ϵ-greedy přístup. Tato dodatečná podmínka přidává další prostor pro zkoušení méně používaných akcí a jejich argumentů.

Pro trénování bylo vytvořeno několik miniher, na kterých se agent mohl zdokonalovat, a které jsou popsány dále.

6.2.1 Minihry

Celkem byly vytvořeny čtyři minihry. Minihry se odehrávají na mapě od DeepMindu, upravené pomocí editoru map, který je k dispozici pro Starcraft. Konkrétně se jedná o úpravu mapy *Flat64*, která byla využita pro vytvoření čtyř různých miniher, na kterých se může každá z podsítí dále trénovat. Těmi jsou:

- *Flat64build* – Cílem je postavit co nejvíce budov. Agent má k dispozici malý počet surovin, které jsou postupně dále těženy jednotkami a dostává odměnu za každou budovu, kterou postaví. Odměna se pohybuje v rozmezí +2 až +10 dle typu budovy. Agent musí vybrat jednotku, která těží a zadat ji příkaz pro stavbu. Pokud nějakou těžební jednotku nepoužívá a nepřiřadí ji zpět k práci, obdrží malou bodovou penalizaci. Časový limit minihry je nastaven na 5 minut.

- *Flat64train* – Cílem je natrénovat co nejvyšší možný počet vojenský jednotek. Na mapě se při spuštění vygenerují náhodně rozmístěné objekty, přičemž agent musí označovat takové budovy, které produkují vojenské jednotky, a ty trénovat. Za označování jiných je malá bodová penalizace, což vede systém k co možná nejrychlejšímu přepínání mezi budovami, ve kterých je možné aktuálně cvičit nějakou jednotku a „neztrácet“ tak čas. K dispozici má určitý předem stanovený počet surovin, které může využít. Časový limit minihry je 7 minut. Agent obdrží různou odměnu v závislosti na typu vytrénované jednotky (+1 až +20).

- *Flat64combat* – Cílem je zničit nepřátelskou základnu. Agent začíná na základně s několika vojáky, které musí dovést na druhou stranu mapy a zničit nepřátelské jednotky a všechny budovy. Obdrží odměnu za každou zničenou nepřátelskou jednotku (+3) a budovu (+2), a naopak obdrží penalizaci za smrt vlastní jednotky nebo budovy (−2). Mapa končí, pokud jsou zničeny všechny vlastní jednotky a budovy, nebo pokud je zničen nepřítel, nebo pokud vyprší časový limit 5 minut.

- *Flat64gather* – Cílem je natěžit co nejvíce surovin. Velmi krátká minihra s časovým limitem 2 minut, která se snaží o maximálně efektivní start bota na začátku hry, se kterým právě systém obtížně bojuje. Agent musí cvičit těžební jednotky (odměna +5), stavět speciální budovy (+50) pro těžbu plynu a natěžit takto co nejvíce surovin. Dále obdrží odměnu za každou natěženou jednotku plynu.

Trénování na minihrách je možné spustit při použití přepínače `--map` a zvolení příslušného názvu mapy, tedy např. `--map Flat64build`. Dále je možné nastavit počet epizod, které má agent odehrát, přepínačem `--episodes`. Faktor učení byl pro všechny sítě nastaven na $3 \cdot 10^{-4}$, faktor entropie na $1 \cdot 10^{-1}$ a jako optimizér byl použit algoritmus *Adam*. Řídící síť nebyla dále samostatně trénována. Během hraní v celé hře dostávala malé odměny, pokud preferovala stavící podsíť v prvních minutách zápasu, a naopak trénovací a bojovou podsíť v těch pozdějších. Jednotlivé podsítě využívají fixní časový interval, po který jsou

aktivní. Po uplynutí tohoto intervalu dojde k dalšímu přepnutí sítě. Pro síť N_B byl tento interval nastaven na 6 sekund, pro N_C na 6 sekund a pro N_T na 15 sekund.

Na každé z těchto miniher byl agent trénován, resp. každá z podsítí na příslušné minihře, dalších zhruba 48 hodin, během kterých si zdokonalil chování v připravených situacích. Tzn. naučil se lépe těžit suroviny, stavět budovy, trénovat jednotky a útočit na nepřítele.

Průběh trénování a nahrané skóre je k dispozici pro každou minihru na grafech 6.9. Pro srovnání je ke každé minihře uvedeno průměrné skóre nahrané lidským hráčem dosahujícího lehce nadprůměrné dovednostní úrovně. Tato hodnota slouží jako výchozí skóre, ke kterému se měl agent přiblížit.

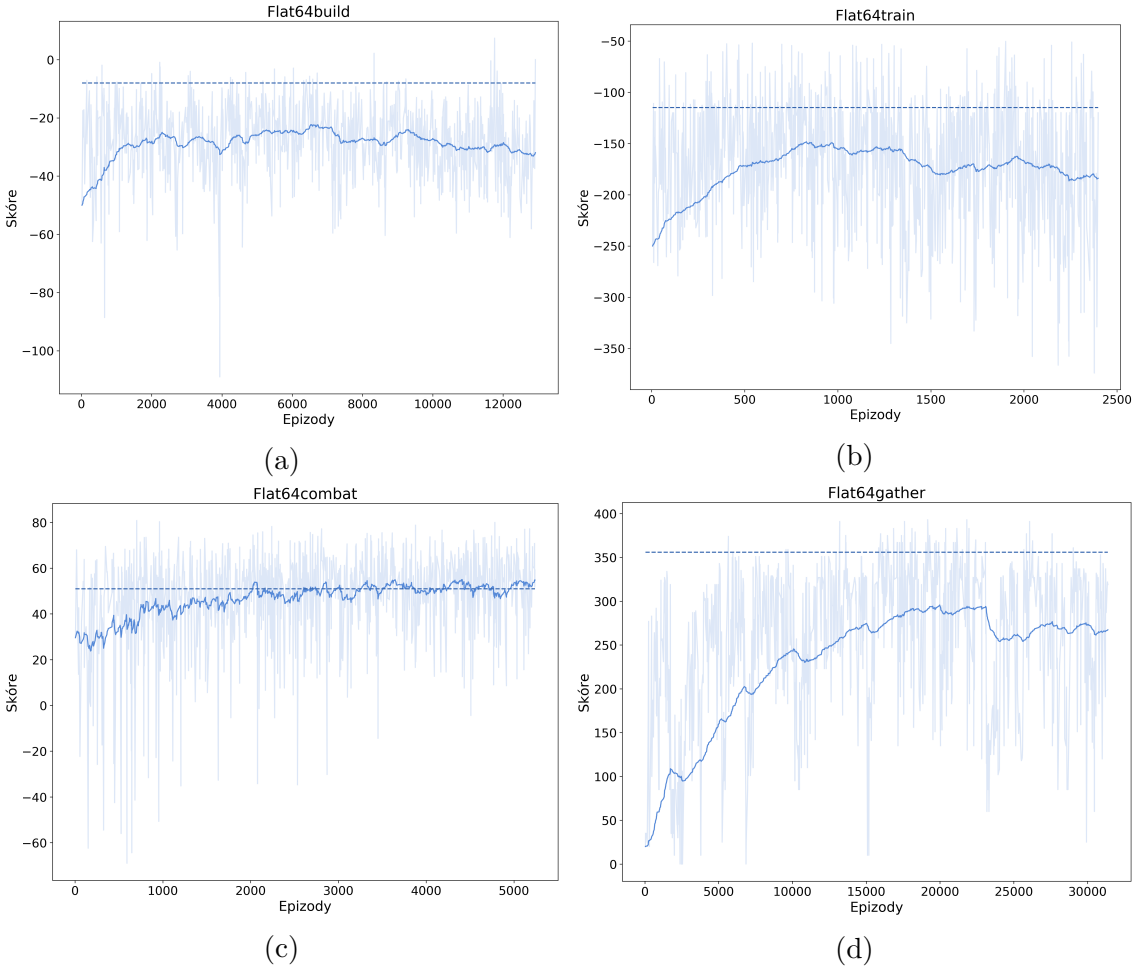
Síť N_B dosahovala průměrného skóre na minimapě *Flat64build* -27 a hráčovo skóre bylo -8. Síť se na naučila rychleji stavět budovy a efektivněji využívat těžební jednotky. Nižší nahrané skóre plyne především z toho, že síť označovala často i jiné jednotky než těžební, za což dostávala bodovou penalizaci. N_T nahrála na minihře *Flat64train* průměrně -167 bodů a hráčské skóre bylo -115. Nižší skóre je obdobné jako u předchozí minihry, jelikož byla síť opět penalizována za označování jiných jednotek a budov, než bylo od ní očekáváno. Po natrénování byla síť schopna trénovat rychleji jednotky, i když jí to stále činí potíže. Síť N_C si vedla velmi dobře na minihře *Flat64combat*, její průměrné skóre dosahovalo 47 bodů a hráčské 51. Naučila se efektivněji používat kameru a bojovat s jednotkami. Největší zlepšení dosáhla síť N_G na minihře *Flat64gather*, naučila se stavět budovy pro těžení suroviny *vespene*, která je velmi důležitá pro start hry. Její průměrné skóre činilo 255, zatímco hráčovo 356. Hráč nahrál skóre vyšší, jelikož byl schopen efektivněji přiřazovat těžební jednotky k surovinám.

6.2.2 Vyhodnocení dotrénovaného modelu

Natrénovaný agent byl opět otestován na mapě *Flat64custom* proti nepřátelskému botovi ve hře nastaveného na nejlehčí úroveň. Oba agenti měli k dispozici jednu oblast pro těžení surovin a začínali na protějších rozích mapy. FoW bylo zapnuto. Agent používal řídící síť a čtyři podsítě se stejnými akcemi jako v podkapitole 6.2.1. Vítězství bylo určeno standardním způsobem, tedy pokud se nepřítel vzdal ještě před veškerým zničením budov a jednotek, což většinou nastává při zničení veškerých bojových jednotek a útoku na základnu, bylo to vyhodnoceno jako výhra. Časový limit byl stanoven na 30 minut, po kterých byla vyhlášena remíza.

Bot odehrál přes 5000 her, během kterých dokázal celkem vyhrát nad protivníkem v 66 % případů a v 6 % případů remizovat. Průměrné skóre během her činilo 5971 bodů, viz grafy 6.10. Díky trénování na minihrách se tedy byl schopen výrazně zlepšit v celkovém hraní. K tomuto přispělo i to, že pouze jedna síť mohla ovládat kameru a ostatní sítě hrály s kamerou fixní. Nicméně se ukázalo síť N_C se byla schopna naučit používat také určitým způsobem kameru, což potvrzují i výsledky z minihry *Flat64combat*. Lze tedy předpokládat, že pokud by byl sítím dán větší prostor pro trénování a delší čas, mohly by být schopny využívat kameru také. Hlavním problémem byla tedy skutečnost, že by se musely ovládání kamery „přeučit“ stejně jako síť N_C, jelikož jsou po naučení ze záznamů touto akcí velmi zkreslené.

Ukázalo se, že samotné záznamy nejsou dostatečné k tomu, aby se neuronové sítě dokázaly naučit ovládat tak komplexní hru jakou je Starcraft. Pro sítě je velmi problematické hledat souvislosti mezi akcemi, které provádějí hráči, jelikož drtivá většina z nich je pohyb kamery anebo operace nicnedělání. Kritické akce, jako je stavění budov či trénování jednotek, dosahují pouze nízkých jednotek procent provedení oproti všem ostatním. Tato



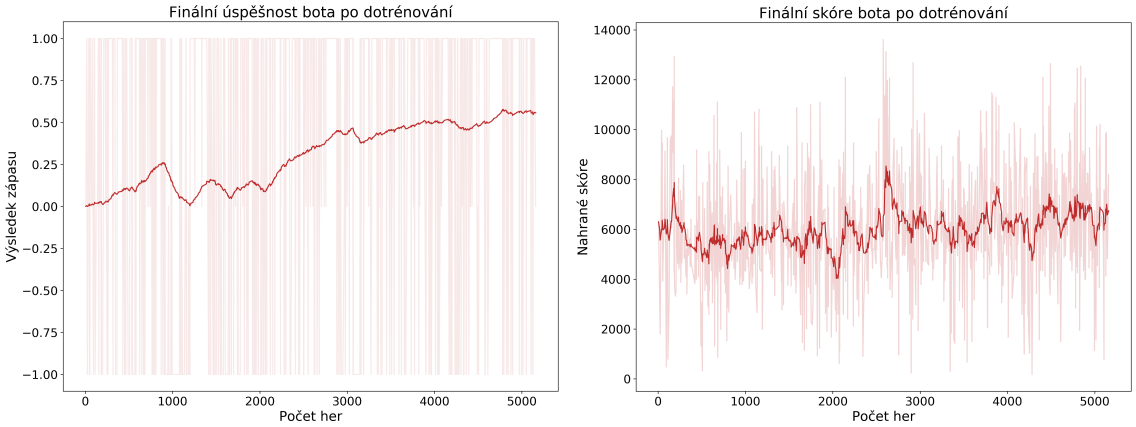
Obrázek 6.9: Průběh trénování na minihrách. Přerušované přímky značí výchozí skóre nahrané hráčem, ke kterým se měly sítě přiblížit. a) Síť N_B na minihře *Flat64build*. b) Síť N_T na minihře *Flat64train*. c) Síť N_C na minihře *Flat64combat*. d) Síť N_G na minihře *Flat64gather*.

skutečnost tak velmi deformuje učení sítí. V samotné hře pak dokážou stavět budovy a jednotky pouze zřídka, což vede k rychlé porážce nepřítelem.

Naopak, model byl schopen pochytit ze záznamů her prostorovou orientaci na obrazovce. Snadněji tedy klikal na budovy se záměrem je označit apod. Co mu však činilo potíže, bylo pochopit systém mapy, kde kamera zabírala pouze její určitou část a vhodným posunem je nutné se po ní pohybovat.

Dále se projevilo, že rozdělení na podproblémy je jedno z možných řešení hraní Starcraftu. K hraní tak není třeba enormně velké neuronové sítě, která by uměla všechno, ale spíše se jeví, že je praktičtější, aby se sítě zabývaly pouze menší částí hry. Jednotlivé sítě jsou schopny naučit se lépe řešit dílčí úkoly, tzn. jedna síť je schopná cvičit jednotky, druhá sbírat suroviny, třetí stavět budovy a čtvrtá útočit, což je logický závěr.

Jedna z hlavních nevýhod, který tento návrh má, je, že chybí tok informací mezi jednotlivými sítěmi. Každá síť je tak schopna hrát sama za sebe, ale netuší nic o dalších sítích nebo jiných aspektech hry. Jsou tak vázány na to, co vykoná síť, která hraje před nimi, a nejsou schopny to nikterak ovlivnit. Tuto nevýhodu měla do jisté míry korigovat řídící



(a) (b)

Obrázek 6.10: Finální výsledky dotrénovaného bota. a) Výsledek zápasů, kde −1 značí prohru, 0 remízu a +1 výhru. b) Dosažené skóre během her.

síť, jejíž cílem bylo naučit se ovládat (aktivovat) jednotlivé podsítě tak, aby dohromady dokázaly hru hrát. Řídící síť však opět nemá možnost přímo ovlivnit některou z podsítí a předat jí nějakým způsobem informace. Nepřímo ji ovlivňuje tak, že ji například aktivuje více nebo méně častěji v různých fázích hry. Je tedy nutné, aby podsítě dokázaly obstojně vykonávat svůj úkol. Mezi sítěmi tak chybí jakási hierarchie informací a záměr, se kterým by sítě mohly hrát a lépe se tak rozhodovat. To je rozhodně jeden z prostorů pro zlepšení.

Je tedy možné říci, že ve srovnání s botem DeepMindu, si vedl navržený bot lépe, i když připustíme fakt, že hrál zjednodušenou hru a na menší mapě. Hlavní problém toho, že bot DeepMindu nedokázal hrát ani proti nejlehčímu skriptovanému botovi ve hře, je totiž ten, že nebyl schopen stavět vhodně budovy, a především rychle trénovat velké množství jednotek a těžit suroviny, což je prakticky základ hraní Starcraftu. Navržený bot se právě snaží tyto aspekty eliminovat tím, že problém rozděluje na dílčí, které je možné snáze řešit a zlepšovat na minihrách.

Pro zrychlení učení je také vhodné definovat lepší odměny pro agenta. Jelikož při použití pouze ternárních odměn za konec zápasu, tj. +1 za výhru, −1 za prohru a 0 za remízu, trvá sítím s klasickými algoritmy posilovaného učení nepřiměřeně dlouho, než se dokáží něco naučit.

Dalším přínosem navrženého bota je, že se snaží hru hrát jako celek, pouze pomocí strojového učení, což je stále neobvyklé, a navazovat tak na výzkum vědců z týmu DeepMind. To, že prakticky neexistují jiní boti, kteří by hráli celou hru pomocí strojového učení, je dáno několika důvody. Většinu existujících botů programují studenti ve svém volném čase a snaží se dosáhnout nějakých výsledků v turnajích. V tomto směru je stále lepší volbou „tvrdé“ programování herních vzorců a jejich maximálně efektivní a přesné využití ve hře. Strojové učení obecně a obzvláště posilované učení je stále ve svých začátcích. Objevují se nové technologie a vylepšují se ty staré. V neposlední řadě je také trénování modelů velmi složité, časově náročné, a tudíž i finančně nákladné, což je pro samostatné jedince velká překážka. Stále se tak jedná o první kroky v této oblasti umělé inteligence, jelikož největším problémem se pro sítě jeví zachycení časových posloupností akcí a hierarchické myšlení s nějakým záměrem a strategií, které je naopak velmi typické pro člověka. Proto jsou lidé v takto komplexních hrách zatím nepřekonatelní.

Kapitola 7

Závěr

Cílem práce bylo vytvořit automatický systém pro hraní strategické hry v reálném čase Starcraft II na základě učení ze záznamů her hráčů. Navržený bot staví na práci týmu DeepMind, který ukázal ve svém článku [35], že Starcraft II je prozatím pro umělou inteligenci velká výzva a využití běžných algoritmů strojového učení nebylo dostatečně účinné na to, aby se jejich systém naučil hru hrát.

Konkrétní záměr byl vytvořit bota, který bude schopen hrát celou hru, i když zjednodušenou. To je relativně neobvyklé v kontextu prvního i druhého dílu Starcraftu, jelikož systémy, které by skrze strojové učení hrály hru jako celek, prakticky neexistují.

Proces návrhu byl rozvržen do dvou etap. První etapa se zabývala učením modelu ze záznamů her pomocí metody učení s učitelem. Jelikož ani DeepMind neuspěl s touto technikou, byly navrženy určité změny, které měly pomoci hraní zlepšit. Model se skládá z několika neuronových sítí, konkrétně z několika herních podsítí, přičemž každá řeší odděleně jiný aspekt hry, např. stavění budov, trénování jednotek a těžení surovin. Mezi těmito podsítěmi pak přepíná hlavní síť a vytváří tak určitou formu hierarchie během hraní. Myšlenkou za tímto návrhem bylo rozdělit složitý problém hraní na dílčí podproblémy.

V první etapě tedy došlo k učení sítí ze záznamů her hráčů. K tomuto účelu byly vybrány vhodné záznamy z dostupných datasetů. Výsledek byl však obdobný, jako v případě DeepMindu, a také očekávaný. Systém nebyl schopen naučit se hrát hru, a to ani v případě zjednodušení a rozdělení hraní na podproblémy. Vedl si jen o trochu lépe, jelikož se mu podařilo vyhrát několik málo her.

Druhá etapa navazuje na předešle naučené sítě, jelikož vykazovaly některé vhodně naučené vlastnosti, například si byly vědomy prostorového uspořádání objektů na obrazovce, což jim umožňovalo snadněji označovat jednotky a budovy. Proto bylo vytvořeno několik miniher, na kterých se mohly jednotlivé sítě dále trénovat a zlepšovat se v řešení svých úkolů. Po dalším natrénování pomocí posilovaného učení byl systém znovu otestován proti skriptovanému botovi nastavenému na nejlehčí úroveň obtížnosti. Za těchto podmínek dokázal s 66% úspěšností nad protivníkem vyhrávat a v 6 % her s ním remizovat. Ukázalo se tedy, že model byl schopen zlepšit se v kritických aspektech hry, tedy především v efektivním těžení surovin, stavěním budov a trénováním vojáků, což vedlo ke zlepšení v celkovém hraní.

Model dosahuje lepší úspěšnosti než bot týmu DeepMind, i když budeme uvažovat fakt, že hrál zjednodušenou hru. Byl totiž schopen naučit se využívat klíčové prvky hry. Stále se však jedná o první kroky v oblasti umělé inteligence pro řešení tak komplexního problému.

Literatura

[1] Abadi, M.; Agarwal, A.; Barham, P.; aj.: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* [online]. 2016, [cit. 2018-01-03]. Dostupné z: https://arxiv.org/pdf/1603.04467.pdf

[2] Babaeizadeh, M.; Frosio, I.; Tyree, S.; aj.: *Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU* [online]. 2016, [cit. 2018-01-10]. Dostupné z: https://arxiv.org/pdf/1611.06256.pdf

[3] Blizzard: *About - StarCraft II WCS* [online]. [cit. 2018-01-08]. Dostupné z: https://wcs.starcraft2.com/en-us/about/

[4] BurnySc2: *MMR Ranges in StarCraft 2* [online]. [cit. 2018-01-14]. Dostupné z: https://burnysc2.github.io/MMRranges/

[5] Buro, M.: Real-Time Strategy Games: A New AI Research Challenge. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* [online]. 2003, [cit. 2018-01-08]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.6742&rep=rep1&type=pdf

[6] Campbell, M.; Hoane, A. J.; Hsu, F.: Deep Blue. *Artificial Intelligence*, ročník 134, č. 1-2, jan 2002: s. 57–83, doi:10.1016/s0004-3702(01)00129-1.

[7] Churchill, D.: *CommandCenter: AI Bot for Broodwar and Starcraft II* [online]. [cit. 2018-01-13]. Dostupné z: https://github.com/davechurchill/commandcenter

[8] Churchill, D.: *SparCraft Home* [online]. [cit. 2018-01-13]. Dostupné z: https://github.com/davechurchill/ualbertabot/wiki/SparCraft-Home

[9] Churchill, D.: *UAlbertaBot: Home* [online]. [cit. 2018-01-13]. Dostupné z: https://github.com/davechurchill/ualbertabot/wiki

[10] Churchill, D.; Saffidine, A.; Buro, M.: *Fast Heuristic Search for RTS Game Combat Scenarios* [online]. [cit. 2018-01-08]. Dostupné z: http://www.cs.mun.ca/~dchurchill/pdf/aiide12-combat.pdf

[11] Clevert, D.-A.; Unterthiner, T.; Hochreiter, S.: *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)* [online]. 2015, [cit. 2018-01-07]. Dostupné z: https://arxiv.org/pdf/1511.07289

[12] Frans, K.; Chen, X.; Abbeel, P.; aj.: *Meta Learning Shared Hierarchies* [online]. 2017 [cit. 2018-01-15].
Dostupné z: https://arxiv.org/pdf/1710.09767.pdf

[13] Glorot, X.; Border, A.; Bengio, Y.: *Deep Sparse Rectifier Neural Networks. Proceedings of Machine Learning Research* [online]. 2011, [cit. 2018-01-13].
Dostupné z: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[14] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep learning*. The MIT Press, 2016, ISBN 02-620-3561-8.

[15] Google: *GOOGLE CLOUD BIG DATA AND MACHINE LEARNING BLOG: An in-depth look at Google's first Tensor Processing Unit (TPU)* [online]. [cit. 2017-12-28].
Dostupné z: https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

[16] Google: *Google Trends: Zájem v průběhu času* [online]. [cit. 2018-01-12].
Dostupné z: https://goo.gl/DmyvLA

[17] Kiseido: *How to play Go* [online]. [cit. 2017-12-27].
Dostupné z: https://www.kiseido.com/ff.htm

[18] Metacritic: *Starcraft for PC Reviews - Metacritic* [online]. [cit. 2018-01-08].
Dostupné z: http://www.metacritic.com/game/pc/starcraft

[19] Metacritic: *Starcraft II: Wings of Liberty for PC Reviews - Metacritic* [online]. [cit. 2018-01-08].
Dostupné z:
http://www.metacritic.com/game/pc/starcraft-ii-wings-of-liberty

[20] Mnih, V.; Badia, A.; Mirza, M.; aj.: *Asynchronous Methods for Deep Reinforcement Learning* [online]. 2016, [cit. 2018-01-07].
Dostupné z: https://arxiv.org/pdf/1602.01783

[21] Mnih, V.; Kavukcuoglu, K.; Silver, D.; aj.: *Playing Atari with Deep Reinforcement Learning* [online]. 19 Dec 2013, [cit. 2018-01-03].
Dostupné z: https://arxiv.org/pdf/1312.5602v1.pdf

[22] Nielsen, M.: *Chapter 6: Deep learning* [online]. [cit. 2018-01-14].
Dostupné z: http://neuralnetworksanddeeplearning.com/chap6.html

[23] Olah, C.: *Understanding LSTM Networks* [online]. [cit. 2018-01-14].
Dostupné z: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[24] Peng, P.; Wen, Y.; Yang, Y.; aj.: *Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games* [online]. 2017, [cit. 2018-01-09].
Dostupné z: https://arxiv.org/pdf/1703.10069.pdf

[25] Polvara, R.; Patacchiola, M.; Sharma, S.; aj.: *Autonomous Quadrotor Landing using Deep Reinforcement Learning* [online]. 2017 [cit. 2018-01-15].
Dostupné z: https://arxiv.org/pdf/1709.03339.pdf

[26] Romstad, T.; Kiiski, J.: *Stockfish: Strong open source chess engine* [online]. [cit. 2017-12-27].
Dostupné z: https://stockfishchess.org/

[27] Schaeffer, J.: A Gamut of Games. *AI MAGAZINE* [online]. 2001, [cit. 2018-01-03].
Dostupné z: https://pdfs.semanticscholar.org/0ccc/33cda90e9dc969d8b72d92ae18c9df23ea5a.pdf

[28] Schulman, J.: *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. Diplomová práce, University of California, Berkeley, 2016.

[29] Schuster, M.; Paliwal, K.: *Bidirectional recurrent neural networks*. 1997, doi:10.1109/78.650093.

[30] Silver, D.; Huang, A.; Maddison, C. J.; aj.: *Mastering the game of Go with deep neural networks and tree search*. *Nature*, ročník 529, č. 7587, jan 2016: s. 484–489, doi:10.1038/nature16961.

[31] Silver, D.; Hubert, T.; Schrittwieser, J.; aj.: *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm* [online]. 2017, [cit. 2018-01-08].
Dostupné z: https://arxiv.org/pdf/1712.01815

[32] Sutton, R. S.; Barto, A. G.: *Reinforcement Learning: An Introduction*. MIT Press, 1998, ISBN 02-621-9398-1.

[33] Sutton, R. S.; Precup, D.; Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, ročník 112, č. 1-2, aug 1999: s. 181–211, doi:10.1016/S0004-3702(99)00052-1.

[34] Tesauro, G.: *Temporal difference learning and TD-Gammon*. mar 1995, doi:10.1145/203330.203343.

[35] Vinyals, O.; Ewalds, T.; Bartunov, S.; aj.: *StarCraft II: A New Challenge for Reinforcement Learning* [online]. 2017, [cit. 2018-01-07].
Dostupné z: https://arxiv.org/pdf/1708.04782

[36] Wikipedia: *Chess* [online]. [cit. 2018-01-14].
Dostupné z: https://en.wikipedia.org/wiki/Chess

[37] Wikipedia: *Go (game)* [online]. [cit. 2018-01-14].
Dostupné z: https://en.wikipedia.org/wiki/Go_(game)

[38] Wikipedia: *Shogi* [online]. [cit. 2018-01-14].
Dostupné z: https://en.wikipedia.org/wiki/Shogi

[39] Yoon, K.: *Differentiable Neural Computer* [online]. [cit. 2018-01-14].
Dostupné z: https://kijungyoon.github.io/DNC/

Příloha A

Obsah přiloženého paměťového média

- *client/* – Obsahuje komprimovanou verzi 4.0.2 klienta Starcraftu II pro Linux.
- *doc/* – Soubory s programovou dokumentací.
- *files/* – Další dodatečné soubory k práci.
- *poster/* – Přiložený plakát k práci.
- *PyBot/* – Zdrojové kódy programu *PyBot*.
- *PyReplayParser/* – Zdrojové kódy programu *PyReplayParser*.
- *thesis/* – Zdrojové soubory pro překlad a výsledné *pdf* s textem práce.
- *videos/* – Některé ukázky z hraní navrženého bota.

Příloha B

Seznam akcí používaných sítěmi

		Učení			Trénování			
ID akce	**Název akce**	N_B	N_T	N_C	N_B	N_T	N_C	N_G
1	move_camera							
2	select_point							
3	select_rect							
6	select_idle_worker							
7	select_army							
39	Build_Armory_screen							
42	Build_Barracks_screen							
43	Build_Bunker_screen							
44	Build_CommandCenter_screen							
50	Build_EngineeringBay_screen							
53	Build_Factory_screen							
56	Build_FusionCore_screen							
64	Build_MissileTurret_screen							
71	Build_Reactor_quick							
79	Build_Refinery_screen							
83	Build_SensorTower_screen							
89	Build_Starport_screen							
91	Build_SupplyDepot_screen							
92	Build_TechLab_quick							
227	Effect_Scan_screen							
239	Effect_SupplyDrop_screen							
300	Morph_Hellbat_quick							
304	Morph_LiberatorAAMode_quick							
305	Morph_LiberatorAGMode_screen							
309	Morph_OrbitalCommand_quick							
312	Morph_PlanetaryFortress_quick							
317	Morph_SiegeMode_quick							
318	Morph_SupplyDepot_Lower_quick							
322	Morph_Unsiege_quick							
326	Morph_VikingAssaultMode_quick							
327	Morph_VikingFighterMode_quick							

		Učení			Trénování			
ID akce	**Název akce**	N_B	N_T	N_C	N_B	N_T	N_C	N_G
355	Research_BattlecruiserWeaponRefit_quick							
361	Research_CombatShield_quick							
362	Research_ConcussiveShells_quick							
371	Research_InfernalPreigniter_quick							
373	Research_MagFieldLaunchers_quick							
406	Research_TerranInfantryArmor_quick							
410	Research_TerranInfantryWeapons_quick							
414	Research_TerranShipWeapons_quick							
419	Research_TerranVehicleAndShipPlating_quick							
423	Research_TerranVehicleWeapons_quick							
451	Smart_screen							
452	Smart_minimap							
459	Train_Banshee_quick							
460	Train_Battlecruiser_quick							
464	Train_Cyclone_quick							
469	Train_Hellbat_quick							
470	Train_Hellion_quick							
475	Train_Liberator_quick							
476	Train_Marauder_quick							
477	Train_Marine_quick							
478	Train_Medivac_quick							
487	Train_Raven_quick							
488	Train_Reaper_quick							
490	Train_SCV_quick							
492	Train_SiegeTank_quick							
496	Train_Thor_quick							
498	Train_VikingFighter_quick							

Tabulka akcí, které měly jednotlivé sítě k dispozici během učení ze záznamů z her a poté během trénování. Akce příslušící sítím jsou barevně vyznačeny.

Příloha C

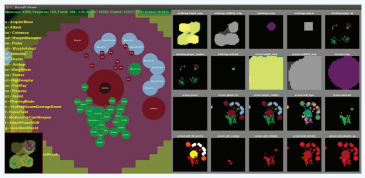
Plakát

Zlepšování systému pro automatické hraní hry Starcraft II v prostředí PySC2

Bc. Jan Krušina
XKRUSI00@STUD.FIT.VUTBR.CZ
VEDOUCÍ: DOC. RNDR. PAVEL SMRŽ, PH.D.

Abstrakt

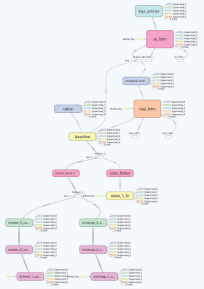
Práce se zabývá vytvořením automatického systému pro hraní strategické hry v reálném čase Starcraft II. Model je trénován ze záznamů her hráčů a dále využívá technik posilovaného učení pro zlepšování vnitřního systému bota. Záměr je vytvořit systém schopný hrát hru jako celek, přičemž staví na frameworku PySC2 pro strojové učení.



Prostředí PySC2, které umožňuje agentovi vidět hru skrze vrstvy rysů (feature layers).

Model

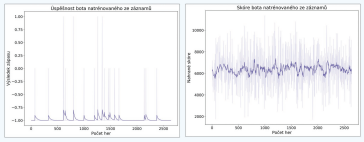
Model agenta se skládá z několika neuronových sítí, přičemž každá z nich řeší určitý aspekt hry a umožňuje tak rozložit složitý problém hraní Starcraftu na dílčí podproblémy. Jednotlivé herní sítě řeší zvlášť stavbu budov, trénování jednotek a útočení na nepřítele. Disponují připravenými sety akcí a hrají částečně zjednodušenou hru. Korigovány jsou pomocí řídící sítě, která mezi nimi přepíná na základě stavu hry.



Model jedné z herních sítí vygenerovaný pomocí nástroje Tensorboard.

Učení a trénování

Proces učení byl rozdělen do dvou fází. Nejprve byly sítě učeny na záznamech z her hráčů. Takto naučený model si osvojil některé dobré vlastnosti, např. prostorovou orientaci na obrazovce.
Testován byl proti zabudované AI ve hře nastavené na nejlehčí úroveň obtížnosti a na zjednodušené mapě.

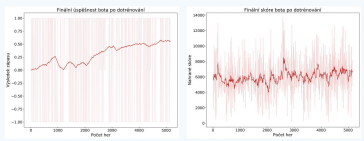


Výsledky bota naučeného ze záznamů z her. Dokázal vyhrát jen několik málo her proti soupeři.

Bot sehrál s nepřítelem více než 2500 her. Nicméně záznamy nestačily na to, aby se bot naučil dobře hrát.

Druhá fáze se tak zaměřila na vylepšení agenta pomocí posilovaného učení. Předučené sítě byly trénovány pomocí algoritmu A3C na několika minihrách, ve kterých se mohly individuálně zlepšovat.

Takto dotrénovaný bot byl znovu otestován proti nepříteli za stejných podmínek a sehrál s ním dalších 5000 her. V těch dosáhnul až 66% úspěšnosti.



Naučený a trénovaný bot byl schopen vyhrát 66 % zápasů.